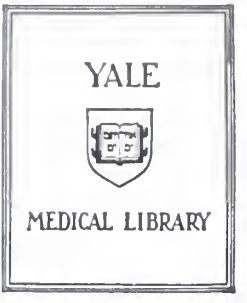
YALE UNIVERSITY LIBRARY
39002010129972

MU: A DOMAIN-INDEPENDENT CASE-BASED EXPERT SYSTEM

MITCHELL JAY SKLAR

1988


YALE
MEDICAL LIBRARY

Digitized by the Internet Archive
in 2017 with funding from
Arcadia Fund

https://archive.org/details/mudomainindepend00skla

YALE MEDICAL LIBRARY
SEP 6 1988

MU: A Domain-Independent Case-Based Expert System

A Thesis Submitted

to the

Yale University School of Medicine

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Medicine

by

Mitchell Jay Sklar

1988

Mech Lib
TH13
.Y12
5651

Abstract

MU: A Domain-Independent Case-Based Expert System

Mitchell Jay Sklar

1988

A medical expert system is a computer program designed to function in the role of a medical consultant. In many such systems described in the literature, the knowledge base of the program is stored in the form of production rules or frames. While these approaches have their advantages, there exist alternative cognitive models as well. We describe MU, an expert system built around a case-based reasoning model. MU's knowledge base is composed of recollections of particular patients or groups of patients, stored flexibly in a format that allows virtually any type of description the user desires. This flexible storage, together with the separation of the database from the mechanics of the inference engine, provides MU with domain independence. MU is also less goal-directed than traditional medical expert systems, relying instead on associational reasoning to produce a varied output of recollections, weighted judgements, conclusions, commentary, and guesses, in a process we term *musing*. Preliminary tests of MU with psychiatric and mammographic cases produced good results, but final evaluation of the system awaits comprehensive trials with larger databases.

Contents

Abstract i

List of Figures iii

Acknowledgements iv

1 Introduction 1
- 1.1 Artificial Intelligence 1
- 1.2 Medical Expert Systems 3
- 1.3 Approaches to Knowledge Engineering 4
- 1.4 Design Goals 14

2 Methods 16

3 Results 21
- 3.1 Psychiatric Cases 21
- 3.2 Mammographic Cases 21

4 Discussion 30
- 4.1 The Mechanics of MU 30
- 4.2 Analysis of Testing 45
- 4.3 Comments 47
- 4.4 Summary 50

Appendix A: Code 51

Appendix B: Clustering Algorithm 79

Appendix C: Evocativeness Algorithm 81

References 83

List of Figures

1	An English translation of a MYCIN production rule.	5
2	A typical frame in PIP.	8
3	A typical manifestations list from INTERNIST	10
4	Example of a psychiatric test case used by MU.	17
5	Example of a radiologic test case used by MU.	20
6	MU's commentary on one of the psychiatric cases.	23
7	Results of psychiatric case testing.	25
8	Table for computing χ^2 value for psychiatric cases	26
9	MU's commentary on one of the mammographic cases.	27
10	Results of mammographic case testing.	28
11	Table for computing χ^2 value for mammographic cases	29
12	A sample point τ used by MU.	31
13	A "prototypical" case for stocking MU.	34
14	A sample of MU's response to a new case.	39
15	Explanatory capability using the *WHY* function.	42
16	The function *DESCRIBE*.	43

Acknowledgements

I deeply appreciate the support and guidance over the last three years from Professor David Gelernter, my advisor in the Department of Computer Science. He graciously accepted his role as mentor and strongly encouraged my interest in artificial intelligence. Thanks also to Scott Fertig, a constant source of helpful advice and comments regarding my project. My advisor in the medical school, Dr. Perry Miller, deserves many thanks for initially encouraging me to pursue this research in the computer science department. I gratefully acknowledge the use of data in this thesis provided by Dr. Chip Swett and Dr. Phyllis Kornguth of the Department of Diagnostic Radiology and Dr. Craig Nelson of the Department of Psychiatry at the School of Medicine.

1 Introduction

1.1 Artificial Intelligence

The last twenty years have brought increasing efforts to explore the applications of computers in medicine. One particularly fertile area has been the field of *artificial intelligence*, which can be broadly described as the branch of computer science that tries to understand and model "intelligent" behavior.

Although experts still debate the exact definition of artificial intelligence, or AI, several important concepts have emerged. For one, AI involves a qualitative departure from ordinary "number crunching"—calculations digesting huge quantities of numerical data. More than just a glorified desk calculator, a computer using AI techniques deals with symbols and abstractions. Consider, for example, the difference between approximating the numerical value for the definite integral:

$$\int_0^{\frac{\pi}{2}} \sin^3 t \, \cos t \, dt \approx 0.25$$

and computing the symbolic value of the indefinite integral:

$$\int^x \sin^3 t \, \cos t \, dt = \frac{\sin^4 x}{4} + C.$$

While the first problem is simply a numerical calculation, the latter cannot be solved without more complex techniques, such as symbolic reasoning, typical of AI. Similarly, the problem of interpreting the meaning of an English paragraph requires an understanding of the significance of words as abstractions, representations of entire ideas. As one paper succinctly explains,

> The term *artificial intelligence* is generally accepted to include those computer applications that involve symbolic inference rather than strictly numerical calculation. [1]

It is precisely this symbolic approach that makes AI such a powerful and flexible tool.

Furthermore, AI programs try to avoid exhaustive searches, relying instead on *heuristics*, or rules-of-thumb. These heuristics are rules and procedures that help a program to make an educated guess about a given situation. In a chess game, for example, to examine all of the possible moves and countermoves from a given board situation would prove computationally infeasible. However, heuristics narrow down the search to a promising few moves that can then be thoroughly explored.

Although most work in AI rests on the twin foundations of symbolic reasoning and heuristics, specific approaches vary greatly. One school of thought emphasizes *cognitive modelling*, imitating the mechanisms of human thought with a computer program. For proponents of this view, AI serves as a valuable tool to help dissect and understand human cognition. They gauge their success by assessing how closely the computer model seems to match the reality of human problem solving. Thus, leaning toward the realm of psychology, Michie defines AI as "the development of a systematic theory of intellectual process." [2]

On the other hand, an alternate tack is to emphasize not the mechanisms of reasoning, but instead the results. This approach recognizes that a computer's hardware and the human brain's "wetware" have deep organizational differ-

ences, and that fast or efficient algorithms for reasoning in one system may be inappropriate in the other. Along these lines, one standard for measuring machine intelligence is the traditional Turing Test [3], in which an observer is allowed to communicate by teletype with a human and a computer, each housed in a separate room. If the observer cannot distinguish between the two, then the computer system is deemed "intelligent." Performance, in this view, is the only standard; the specific methods of reasoning are irrelevant.

Most researchers, of course, would choose an approach between these two extremes. Models of human thought serve as a guide for constructing artificial intelligence systems, but performance remains the benchmark for measuring success. More detailed discussions of the broad field of artificial intelligence can be found in several texts [4,5,6].

1.2 Medical Expert Systems

Traditional problems in artificial intelligence include the recognition of objects in a visual scene, the understanding of a "natural language" such as written English, and voice recognition. Another historical field of interest has been the development of *expert systems*, computer programs designed to emulate human expertise in a particular area. These programs function much like human consultants offering advice. The field of medical diagnosis has proved to be especially well suited to attack by these methods, posing interesting problems both in constructing cognitive models of medical experts and building systems that can perform adequately and arrive at the correct diagnosis. In addition, medical

diagnosis is a sufficiently complex and important problem that a functioning medical expert system would be a valuable achievement.

Like a human consultant, an expert system cannot rely merely on the rules of logic alone. Neither will general strategies of reasoning, such as dividing a goal into several smaller subgoals, suffice. Instead, an expert needs specialized information in a particular area. For example, a medical expert, whether human or computer, would be unlikely to diagnose a patient as having sickle cell anemia if unfamiliar with the natural history, signs and symptoms, or pathophysiology of that particular hemoglobinopathy. (In fact, sickle cell anemia went unrecognized for many years [7,8].) Experts require a knowledge base and that knowledge base must necessarily be specific to a given domain or area of expertise.

1.3 Approaches to Knowledge Engineering

Constructing such a knowledge base, however, provides several obstacles to surmount. What is the best method to acquire expert knowledge? In what form should that knowledge be structured and stored? How can that knowledge be pieced together to arrive at a valid conclusion? It is not immediately obvious how human experts deal with these issues of *knowledge acquisition*, *knowledge representation*, and *knowledge use*; and, as mentioned previously, the human approach may not be suitable for computer expert systems. Researchers in the field refer to these issues collectively as *knowledge engineering* [1], one of the dominant directions for study in medical expert systems.

It is useful to review some of the strategies utilized by medical expert systems in the literature. One large class of systems, epitomized by the MYCIN system developed at Stanford for diagnosing bacteremias [9], stores expert knowledge in the form of "production rules," which can be thought of as simple if-then rules. A English translation of such a rule [10] is shown in Figure 1. Breaking

---

IF The gram-stain of the organism is gram-positive
AND the morphology of the organism is coccus
AND the growth conformation of the organism is chains

THEN conclude that the identity of the organism is streptococcus
(modifier: the certainty tally for the premise times .7)

Figure 1: An English translation of a MYCIN production rule [10].

---

up the knowledge base into these small if-then chunks gives the system a high degree of inherent modularity. Human experts can slowly revise the knowledge base by adding new rules, allowing the system to grow incrementally. Because the data relations are not deeply embedded within a data structure such as a decision tree, errors within the knowledge base are more easily detected and corrected. Furthermore, the modularity allows parts of the knowledge base to be shared by systems with overlapping areas of expertise. A final advantage is the inherent mechanism for explanation: in order to follow the path of the system's reasoning, the user merely tracks the string of logical rules leading to the final conclusion. Thus the system, unlike the classic example of a Bayesian

statistical system, does not suffer from being perceived as a "black-box" with obscure inner workings.

It is not clear, however, that a rule-based system provides an accurate cognitive model for human expert diagnosis. The data available to an expert clinician is often incomplete or partially inaccurate. Particular items of data may need to be ignored by the expert because they contradict other facts, while other seemingly trivial information may prove crucial to the final diagnosis. Intuitively it would seem that an expert clinician does not construct a formal logical chain, but instead relies on a cycle of hypothesis generation and testing, with continual refinement as more data becomes available to suggest or strengthen a diagnosis. As one paper states:

> Our study clearly illuminates an important difference between the expert in practice and the expert as often pictured in literature or folklore. The epitome of the expert in fiction is the detective who, through superior deductive powers and by sheer force of logic, organizes the facts at hand in such a way that they lead to a single, inevitable conclusion. By contrast, the real-world clinician seems to rely much more heavily upon "guessing," the initial hypothesis typically being based on precious little data. These guesses are apparently prompted by patterns of clinical findings or by specific complaints that bring to mind particular diseases. They physician then tries to demonstrate the correctness of his or her guesses, moving to new hypotheses only if the initial impressions prove untenable. [11]

Indeed, the paradigm of forming a logical chain of conclusions may be the strategy used by inexperienced clinicians such as medical students, with expert clinicians depending instead on learned associations between symptoms and their associated diseases or suggested tests for further investigation.

> This view is corroborated, in part, by the observed differences between the diagnostic approach of a medical student or newly minted

> doctor and that of a practicing expert. The novice struggles "from first principles" initially to propose plausible theories and then to rule out unlikely ones, whereas the expert simply recognizes the situation and knows the appropriate response. [12]

The expert clinician eliminates the intermediate steps in if-then rule chaining, apparently basing his hypotheses on compiled associations. When asked to explain the line of reasoning to a novice like a medical student, however, the expert will often resort to constructing the logical chain, a process that takes considerably more time and probably does not accurately reflect his own diagnostic thought processes.

Another drawback to rule-based expert systems is that the modularity of the knowledge base encourages a simplistic, reductionist view of the domain. Without a data structure to synthesize a global view of each disease, it is more difficult to discover new associations between diseases or to recognize that two diseases are, in fact, merely differing manifestations of the same underlying pathology. Similarly, a rule-based design would make it difficult to postulate that a heterogeneous disease classification ought to be broken down into smaller, more consistent, categories.

Some of these problems can be avoided by medical expert systems that construct their knowledge base on the concept of *frames* [13], such as PIP, a program for obtaining a history of the present illness [12]. A frame is a structured collection of closely related facts, stored in "slots," about a given disease or physiologic state. The frame related to the nephrotic syndrome, excerpted in Figure 2, has slots filled with facts about associated physical and laboratory

```
NAME: nephrotic syndrome
IS-A-TYPE-OF: clinical state
FINDING: low serum albumin concentration
FINDING: heavy proteinuria
FINDING: > 5 grams/24hrs proteinuria
FINDING: massive symmetrical edema
FINDING: either facial or peri-orbital and symmetrical edema
FINDING: high serum cholesterol concentration
FINDING: urine lipids present
MUST-NOT-HAVE: proteinuria absent
IS-SUFFICIENT: both massive pedal edema
   and > 5grams/24hrs proteinuria
⋮
MAY-BE-CAUSED-BY:
   acute glomerulonephritis
   chronic glomerulonephritis
   nephrotoxic drugs
   insect bite
   idiopathic nephrotic syndrome
   systemic lupus erythematosis
   diabetes mellitus
MAY-BE-COMPLICATED-BY:
   hypovolemia
   cellulitis
MAY-BE-CAUSE-OF:
   sodium retention
DIFFERENTIAL DIAGNOSIS:
   if neck veins elevated, consider:
      CONSTRICTIVE PERICARDITIS
   if ascites present, consider:
      CIRRHOSIS
   if pulmonary emboli present, consider:
      RENAL VEIN THROMBOSIS
```

Figure 2: A typical frame in PIP (excerpted from [12]).

findings, specific criteria for determining if the nephrotic syndrome is indeed present, and the various possible causes and complications. Since PIP has a conceptual representation of the nephrotic syndrome in its entirety, it can avoid manipulating intermediate-level bits and pieces of information. There is no need to know a specific physiologic rule such as "IF the serum albumin is low, THEN the plasma oncotic pressure will be decreased," because the higher-level associations between the nephrotic syndrome and periorbital edema have already been stored. The frame-based classification scheme also allows an expert system to respond to broad queries such as "Please describe the nephrotic syndrome," and may provide some hints as to the organizational schemes of a human expert's mind.

The approach taken by INTERNIST [14] is similar in that the knowledge base is partitioned into disease states. INTERNIST, however, concentrates intensively on the associations between a disease and its signs and symptoms, giving relatively less attention to the other information that PIP included such as etiology, complications, and differential diagnosis lists. A disease is described simply by a list of its associated manifestations along with the strengths of those associations. An abbreviated manifestation list is shown in Figure 3. For each manifestation in the list, the first number represents the "evoking strength": how strongly that particular manifestation brings to mind the diagnosis in question, on a scale of zero to five. This is qualitatively similar to the statistical concept of positive predictive value. The second number is a frequency measure, representing how

ALCOHOLIC HEPATITIS

age 16 to 25...0 1
age 26 to 55...0 3
age gtr than 55...0 2
alcohol ingestion recent hx...2 4
alcoholism chronic hx...2 4
sex female...0 2
sex male...0 4
urine dark hx...1 3
weight loss gtr than 10 percent...0 3
abdomen pain acute...1 2
abdomen pain colicky...1 1
abdomen pain epigastrium...1 2
abdomen pain non-colicky...1 2
abdomen pain right upper quadrant...1 3
anorexia...0 4
diarrhea acute...1 2
myalgia...0 3
vomiting recent...0 4
⋮
liver biopsy periportal infiltration neutrophils...3 5
liver biopsy periportal infiltration round cells...1 2
liver biopsy small bile ducts prominent...1 2

LINKS FOR ALCOHOLIC HEPATITIS:

PREDISPOSES TO	mallory-weiss syndrome...1 1
CAUSES	sinusoidal or postsinusoidal portal hypertension...1 1
CAUSES	hepatic encephalopathy...2 2
CAUSES	renal failure secondary to liver disease <hepatorenal syndrome>...2 2
COINCIDENT WITH	pancreatitis acute...2 2
PRECEDES	micronodal cirrhosis <laennec's>...2 3

Figure 3: Part of a typical manifestations list from INTERNIST [14]. The first number after each manifestation is its evoking strength for the diagnosis; the second is the frequency of the manifestation in the disease (each on a scale of zero to five).

often that manifestation is present in a given disease, also on a scale of zero to five. We can think of this as the conditional probability of finding a given sign or symptom given a particular disease. In effect these associational strengths are quite similar to traditional Bayesian statistics, although INTERNIST is much more flexible in its analysis than a strict Bayesian system. This paradigm allows the program to perform quite well when presented with fact-laden data, even with rather difficult tests such as the clinicopathological cases from the *New England Journal of Medicine* [14]. The actual construction of the manifestation lists, however, presents a difficult and time-consuming task. Building the knowledge base of DXPLAIN [15], a system with an internal structure much like INTERNIST, involved some 65,000 relationships between diseases and their signs and symptoms. Most of these associational strengths simply do not exist in the literature and must be estimated empirically by domain experts.

An alternate approach, one that forms the foundations for our own system, MU, is to build a medical expert system around a database of specific patient cases. The underlying cognitive model has a strong intuitive appeal. As any third-year medical student can attest, gaining clinical expertise depends not so much on "book learning," but rather on the examination of hundreds or thousands of patients. In the hospital, one frequently hears a clinician suggesting, "You know, that reminds me of a case I saw during my fellowship training. That patient turned out to have disease x; perhaps this patient has the same diagnosis." Searching a large database of cases provides a measure of flexibility,

especially when diagnosing a patient whose disease does not "follow the rules" or does not fit neatly into a typical diagnostic frame.

Several researchers have investigated this case-based cognitive model, together with the implications for associational chaining of memories. This idea is not new—over forty years ago one author wrote of building a "memex," a machine to serve as a memory aid, based on such a model of the human mind:

> The human mind ...operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain. It has other characteristics, of course; trails that are not frequently followed are prone to fade, items are not fully permanent, memory is transitory. Yet the speed of action, the intricacy of trails, the detail of mental pictures, is awe-inspiring beyond all else in nature ... [With the aid of the "memex," the] physician, puzzled by a patient's reactions, strikes the trail established in studying an earlier similar case, and runs rapidly through analogous case histories ... [16]

An early system built by Feinstein and coworkers at Yale [17,18] involved using a database of clinical information on several hundred patients with primary lung cancer, with the goal of predicting survival rates. Their approach was to estimate the probability of survival of a presenting patient by culling from the database a subset of patients with matching clinical features. Survival could then be estimated from these historical controls. A similar system, in the domain of ischemic heart disease, was developed by Rosati et. al. [19], using a databank of some 3,000 patients to predict prognosis and plan therapy. These programs, as well as others [20,21], emphasize using the database as a source for compiling statistics *dynamically*. They resemble a Bayesian statistical system, but differ in that the conditional probabilities are computed on the

fly, recalculated from the ever-growing database with each query about diagnosis, prognosis or treatment. Systems built upon a large database can provide impressive clinical accuracy and utility. [22,23,24]

A diagnostic system described by Kulikowski [25] avoids the method of compiling statistical summaries, using instead the mathematical technique of linear transformation to classify a presenting case into a diagnosis group among the cases in memory. While this transformation improves the separation into diagnostic groups, the system loses its transparency to the user and dispenses its diagnoses without any comprehensible explanation of its reasoning. Such a system, as accurate as it may be, cannot claim to model human thought.

The case-based reasoning systems developed by Stanfill and Waltz [26] and Kolodner [27] are more similar to the approach of our program MU. Both emphasize the value of *experience* in arriving at a diagnosis. The output from these programs, however, is limited in scope. In constructing MU we attempted to use the case-based model as a set of girders supporting a more elaborate structure, called *musing* [28,29,30]. More than just a goal-directed approach aimed at determining the final diagnosis, musing refers to a running series of commentaries and memory associations, modelled after the heterogeneous comments that might come from a human expert. A presenting case might set off a chain of associations, or prompt the expert to give more weight to certain features of a case, or to focus on certain atypical aspects of a case; all within the context of the expert's personal experience. The information derived during the *process* of

looking for a solution can be as significant as the solution itself. The importance of such commentary in place of solely "answer-directed" medical expert systems has been emphasized in the critiquing systems of Miller [31,32,33] and in other work [34].

1.4 Design Goals

With this background in mind, our aim was to construct MU, a case-based medical expert system incorporating the concept of musing. When presented with a new patient case, the system would have the opportunity to provide a running commentary of its thought processes: the cases it recollects, the weighting it assigns to features in the presenting case, chains of associations and conclusions derived from the database, surprising aspects in the presenting case, guesses for the diagnosis. Such varied output might prove valuable to the user.

Moreover, we designed flexibility into the database as well. MU is domain-independent; that is, it can handle problems in one area (such as psychiatric diagnosis) just as well as problems in a completely unrelated field (such as mammographic diagnosis). With a large enough database, the system could expand the differential diagnosis for a difficult case by recollecting specific cases of unusual diseases. New associations between diseases or new clinical syndromes might be uncovered as a user interacts with the system.

Finally, MU was constructed with built-in explanatory facilities, so that a user could understand the trail of the system's musing. As some authors have

stressed [35], a clinician is much more likely to utilize a system with comprehensible explanations.

2 Methods

In order to implement the design goals described above, a working version of our case-based expert system, called MU, was constructed by the author. The system was written in T [36,37], a programming language developed at the Yale University Department of Computer Science, related to the language LISP [38] and a LISP dialect called SCHEME [39]. The final version of MU consisted of more than 1500 lines of computer code divided among some 140 procedures. The full text of the code is included in Appendix A. The system currently runs on SUN desktop workstations with a multi-window bitmapped display and a mouse pointer.

As the system was being developed, we constructed test databases in several different domains, both medical and non-medical in nature. Samples of selected output are included in the Results and Discussion sections. In order to measure the performance of the system more objectively, however, we wished to test the system with cases provided by an outside expert, thus avoiding the introduction of any unintentional bias in case selection. We also wanted to evaluate the performance of the system on actual patient data.

To this end we obtained sample medical databases from two independent sources. One source was Dr. Craig Nelson of the Department of Psychiatry at Yale, who kindly made available data on fifty-two patients with a history of depressive illness. A typical case, in the list format of T, is shown in Figure 4. The data consisted primarily of the results of a modified Hamilton depression

```
((patient-id 14)
(age 22)
(sex female)
(dmi-level 34)
(response some-response)
(clinical-diagnosis doubtfully-melancholic)
(depressed-mood 3)
(feelings-of-guilt 2)
(suicide 2)
(insomnia-early 2)
(insomnia-middle 2)
(insomnia-late 1)
(work-and-activities 3)
(retardation 1)
(agitation 0)
(anxiety-psychic 2)
(anxiety-somatic 2)
(appetite 1)
(loss-of-energy 2)
(genital-symptoms 0)
(hypochondriasis 0)
(loss-of-weight 0)
(insight 0)
(diurnal-variation-morning 0)
(diurnal-variation-evening 0)
(depersonalization 2)
(paranoid-symptoms 1)
(obsessional-compulsive 0)
(helplessness 2)
(hopelessness 2)
(worthlessness 2)
(ruminative-thinking 0)
(decreased-concentration 1)
(ability-to-experience-pleasure 3)
(delusional-thinking 0)
(hypo-activity 1)
(lack-of-responsiveness 0)
(distinct-quality 2)
(global-severity 4))
```

Figure 4: Example of a psychiatric test case used by MU. The possible values for `response`, indicating the response to the anti-depressant medication desipramine, are: responder, some-response. The possible values for `clinical-diagnosis` are not melancholic, doubtfully melancholic, probably melancholic, definitely melancholic.

scale, a questionnaire on the self-described severity of depressive symptomatology [40,41]. Each attribute in the Hamilton scale consists of a symptom, such as `feelings-of-guilt`, and its associated value, usually a numerical rating on a scale from 0 to 4 or from 0 to 5. Some of the cases contained incomplete data from the rating scale, with features missing. In addition to the modified Hamilton ratings, there was information regarding the clinical diagnosis of the type of depression as determined by a psychiatrist, ranging along a spectrum from not-melancholic (a milder form of depression) to definitely-melancholic (a more severe form). Most cases also contained data relating to the patient's observed clinical response to a particular anti-depressant medication, desipramine (DMI), as well as measured blood levels of that drug.

The fifty-two cases were randomized and divided into three groups. The first group, consisting of 20 cases, provided the initial stock for the system's memory. During the construction of MU, the second group, consisting of 12 cases, was used to test and refine the program routines as they were being developed. The third group of psychiatric cases, 20 patients, was not used at all during the development phase; instead, these cases were reserved for a final test of the completed system. MU was never presented with these last 20 cases until the final testing.

For the testing, the system was stocked with all of the 32 psychiatric cases that had been used in the development phase. The remaining 20 cases were stripped of the features relating to response to desipramine and clinical diagnosis

(some of the cases were incomplete and missing that information). Each of the 20 cases was entered individually, with the system directed to the goal of guessing the response to the anti-depressant. After each case, MU was also directed to guess the clinical diagnosis.

Our second test of the system was in the domain of radiology, specifically mammography. Dr. Chip Swett and Dr. Phyllis Kornguth of the Department of Diagnostic Radiology at Yale generously provided data from 31 mammographic exams. A sample case is displayed in Figure 5. The system was fully completed before any of these cases were used. None of the cases was used during the development phase of the program, thereby eliminating any bias in constructing the system to perform well on these specific cases.

The thirty-one radiology cases were randomized into two groups. The first group, 21 cases, was used to stock the system's memory. The second group, 10 cases, was stripped of information regarding the clinical diagnosis and presented as a test to the system.

Data from the system's guesses, both in the psychiatric domain and the radiologic domain, were analyzed for statistical significance with the χ^2 test with the Yates correction for small expected values [42].

```
((ID 19)                    ; identification number for patient
 (AGE 58)                   ; age in years of patient
 (DEN IS)                   ; density of lesion (high or isodense)
 (BORD-COMP N)              ; is the complete border seen?
 (TYPE IRR)                 ; border type (smooth, irreg, lobulated)
 (DEF N)                    ; is the border well-defined?
 (LOC UPPER INNER)          ; location in breast which quadrant
 (SIDE L)                   ; left or right breast
 (SZ-CH Y)                  ; size change since last film
 (DNS-CH N)                 ; density change since last film
 (COM N)                    ; comet (a particular film finding)
 (HALO N)                   ; halo (a particular film finding)
 (BACK MOD)                 ; background of breast tissue
                            ;   (fatty, moderate, dense)
 (MCAN N)                   ; microcalcification number (none, many)
 (MCATY N)                  ; microcalcification type (none, fine)
 (LG N)                     ; large calcification
 (ARCH N)                   ; architectural distortion
 (DUCP N)                   ; ductal prominence
 (MSOTHR N)                 ; other mass
                            ;   (ipsilateral or contralateral)
 (PAL N)                    ; other masses palpable?
 (SZ-PA N)                  ; palpable size larger than film image?
 (SK N)                     ; skin changes (none, retraction, nipple)
 (NI N)                     ; nipple inversion
 (AD N)                     ; adenopathy
 (FA S)                     ; family history (none, sister, mother)
 (PE N)                     ; personal history of cancer
 (DIAGNOSIS CA-INF-DUCT); pathologic diagnosis
 )
```

Figure 5: Example of a radiologic test case used by MU. The meaning of each of the abbreviations is given in the comment column on the right. The values for diagnosis included infiltrating ductal carcinoma, infiltrating lobular carcinoma, colloid carcinoma, carcinoma (unspecified type), metastatic melanoma, metastatic adenocarcinoma in lymph node, papilloma, fibroademona, cyst, and plasmacytoma.

3 Results

3.1 Psychiatric Cases

A sample of the commentary that MU provides when presented with a psychiatric test case is given in Figure 6. Runtime for this case was approximately 30 seconds. The interpretation and meaning of MU's output is explained in the Discussion section. Figure 7 contains the results of the performance of the system applied to the psychiatric cases. As shown, for the 16 cases where the response to desipramine was known, the system correctly guessed the response 9 times (56% correct). Analysis by the χ^2 test (see Figure 8) showed that the guesses for clinical response were not significantly different from random guessing. When guessing the clinical diagnosis, however, the system ascertained the correct choice (out of four possibilities) in 10 out of the 20 test cases (50% correct). For guessing clinical diagnosis, the χ^2 test showed that the system did indeed perform significantly better than random guessing ($p < 0.10$).

3.2 Mammographic Cases

An example of a test run with a mammographic case is shown in Figure 9. Runtime for this simpler case was approximately 10 seconds. The table in Figure 10 shows the performance with the mammographic cases. Of the 10 cases tested, the system correctly guessed the exact diagnosis in 6 cases (60%). In two more cases (20%) , the system was partially correct in its diagnosis. One of these was a case where MU guessed carcinoma (unspecified type), while the

correct diagnosis was infiltrating ductal carcinoma, the most common type of breast carcinoma, accounting for some seventy-six percent of all breast carcinomas [43]. In the other partially correct case, MU guessed infiltrating lobular carcinoma while the actual diagnosis was given as carcinoma (unspecified type). In two of the cases (20%), the guesses were incorrect. One was a case of infiltrating ductal carcinoma misdiagnosed as a fibroadenoma, and one was a case of plasmacytoma. Plasmacytoma, a rare tumor that can occur in the breast, was not represented by any cases already stored in the database. Analysis by χ^2 (see Figure 11) showed that these results of guessing the pathologic diagnosis were also statistically significant at a level of $p < 0.01$.

```
> (query
'( (patient-id 7)                    (age 23)
(sex male)                           (dmi-level 123)
(depressed-mood 2)                   (feelings-of-guilt 1)
(suicide 2)                          (insomnia-early 1)
(insomnia-middle 1)                  (insomnia-late 0)
(work-and-activities 3)              (retardation 0)
(agitation 0)                        (anxiety-psychic 4)
(anxiety-somatic 2)                  (appetite 1)
(loss-of-energy 2)                   (genital-symptoms 2)
(hypochondriasis 3)                  (loss-of-weight 0)
(insight 1)                          (diurnal-variation-morning 2)
(diurnal-variation-evening 0)        (depersonalization 0)
(paranoid-symptoms 0)                (obsessional-compulsive 0)
(helplessness 2)                     (hopelessness 2)
(worthlessness 2) ))

Evocativeness: 2.34

I am reminded of the following cases:

Case Number     PATIENT-ID              RESPONSE
-----------     --------------------    --------------------
29              (56)                    SOME-RESPONSE
0               (21)
31              (14)                    SOME-RESPONSE
28              (62)                    SOME-RESPONSE
24              (65)
2               (26)
21              (57)                    RESPONDER
8               (91)                    SOME-RESPONSE
11              (12)                    SOME-RESPONSE
30              (44)                    SOME-RESPONSE

⋮
```

Figure 6: MU's commentary on one of the psychiatric cases (continued on the next page). Note the mix of cases brought to mind, evocativeness, surprises, conclusions, and guesses. These are explained further in the discussion section. For this particular case, the response to desipramine was unknown. MU correctly guessed the actual clinical diagnosis, definitely melancholic.

```
:

I'm surprised.
I would have predicted that the HYPOCHONDRIASIS would be (0).
I'm surprised.
I would have predicted that the INSIGHT would be (0).

Conclusions:
  DELUSIONAL-THINKING            (0)

Evocativeness: 1.26

I am reminded of the following cases:

Case Number    PATIENT-ID              RESPONSE
-----------    --------------------    --------------------
0              (21)
1              (51)                    RESPONDER
2              (26)
3              (80)                    SOME-RESPONSE
4              (74)                    SOME-RESPONSE
6              (71)                    RESPONDER
7              (61)
8              (91)                    SOME-RESPONSE
11             (12)                    SOME-RESPONSE
12             (23)                    SOME-RESPONSE
13             (18)                    SOME-RESPONSE
14             (67)                    SOME-RESPONSE
16             (11)                    SOME-RESPONSE
17             (72)                    RESPONDER
21             (57)                    RESPONDER
22             (84)                    RESPONDER
24             (65)
25             (86)                    SOME-RESPONSE
27             (15)                    SOME-RESPONSE
28             (62)                    SOME-RESPONSE
29             (56)                    SOME-RESPONSE
30             (44)                    SOME-RESPONSE
31             (14)                    SOME-RESPONSE

My best guess for RESPONSE is SOME-RESPONSE

> (guess 'clinical-diagnosis)
My best guess for CLINICAL-DIAGNOSIS is DOUBTFULLY-MELANCHOLIC
```

Figure 6 (continued).

Psychiatric Test Cases						
	Response to Desipramine			Clinical Diagnosis–Melancholic		
ID	Guess	Actual	Correct?	Guess	Actual	Correct?
2	some	some	+	definitely	definitely	+
9	some	some	+	probably	definitely	−
7	some	?	?	doubtfully	doubtfully	+
40	some	responder	−	probably	probably	+
47	some	?	?	probably	not	−
3	some	responder	−	probably	probably	+
6	some	some	+	definitely	definitely	+
16	some	some	+	definitely	definitely	+
73	some	some	+	probably	definitely	−
59	some	?	?	probably	not	−
42	responder	some	−	definitely	definitely	+
27	some	some	+	definitely	definitely	+
13	some	some	+	definitely	probably	−
5	responder	responder	+	definitely	probably	−
20	some	responder	−	definitely	probably	−
82	some	some	+	definitely	probably	−
90	some	responder	−	definitely	definitely	+
52	some	?	?	definitely	not	−
39	some	responder	−	probably	definitely	−
70	responder	some	−	definitely	definitely	+

Figure 7: Results of psychiatric case testing. The symbol + shows a correct guess, the symbol − marks an incorrect guess, and the symbol ? marks a case where the actual value of the response or clinical diagnosis was unknown. MU's guesses were correct for 9 out of 16 known values for the response to desipramine (56%), and for 10 out of 20 known values for the clinical diagnosis (50%).

Psychiatric Cases Data Analysis

Response	number in database	p_i	n_i	E_i
responder	10	0.357	6	2.143
some-response	18	0.643	10	6.429
Total	28	1.000	16	8.571

Expected number of correct answers: 8.571
Actual number of correct answers: 9
Expected number of incorrect answers: 7.429
Actual number of incorrect answers: 7
χ^2 value (with Yate's correction): 0.001 (p not significant)

Diagnosis	number in database	p_i	n_i	E_i
not-melancholic	3	0.094	3	0.281
doubtfully-melancholic	8	0.250	1	0.250
probably-melancholic	11	0.343	6	2.063
definitely-melancholic	10	0.313	10	3.125
Total	32	1.000	20	5.719

Expected number of correct answers: 5.719
Actual number of correct answers: 10
Expected number of incorrect answers: 14.281
Actual number of incorrect answers: 10
χ^2 value (with Yate's correction): 3.5 ($p < 0.10$)

Figure 8: Tables for computing χ^2 value for psychiatric cases. The number p_i represents the frequency of a value i in the initial database, and n_i is the number of times the value occurs among the test cases. The last column, E_i, is the expected number of correct guesses for value i if the system guessed randomly, is computed from $p_i n_i$. For example, there were 6 test cases with the diagnosis of probably-melancholic. If the system guessed purely based on the diagnosis frequencies already in the database, it would be correct $(0.343)(6) = 2.063$ times for the diagnosis of probably-melancholic. Overall, the total number of correct guesses for clinical-diagnosis was 10, compared to an expected number of 5.719 ($p < 0.10$). The total number of correct guesses for desipramine response was 9, compared to an expected number of 8.571 (p not significant).

```
> (query '(
(id 30)              (age 42)       (den hi)
(bord-comp n)        (type lob)     (def y)
(loc lower inner)    (dns-ch n)     (com y)
(halo n)             (back fty)     (mcan n)
(mcaty n)            (lg n)         (arch n)
(ducp n)             (msothr n)     (pal n)
(sz-pa same)         (sk n)         (ni n)
(ad n)               (fa n)         (pe my) ))

Evocativeness: 2.16

I am reminded of the following cases:

Case Number    ID                    DIAGNOSIS
-----------    --------------------  --------------------
12             (6)                   CA-COLLOID
6              (15)                  FA
4              (16)                  MET-MELANOMA
8              (7)                   FA
9              (21)                  CYST
13             (14)                  CA-INF-DUCTAL
1              (12)                  FA
20             (24)                  FA

I'm surprised.  I would have predicted that the COM would be (N).

My best guess for DIAGNOSIS is FA
```

Figure 9: MU's commentary on one of the mammographic cases. The system is surprised at the radiologic finding of a comet, and guesses (correctly) that the diagnosis is fibroadenoma despite having recalled some more malignant lesions as well.

Mammographic Test Cases			
	Diagnosis		
ID	Guess	Actual	Correct?
11	fibroadenoma	ca-inf-ductal	−
25	fibroadenoma	fibroadenoma	+
2	carcinoma	carcinoma	+
22	fibroadenoma	fibroadenoma	+
26	carcinoma	ca-inf-ductal	±
32	ca-inf-lobular	carcinoma	±
3	carcinoma	carcinoma	+
29	fibroadenoma	fibroadenoma	+
4	fibroadenoma	fibroadenoma	+
30	fibroadenoma	plasmacytoma	−

Figure 10: Results of mammographic case testing. The symbol + shows a correct guess, the symbol − marks an incorrect guess, and the symbol ± marks a partially correct guess. MU's guesses were correct for 6 out of 10 of the cases (60%), and partially correct in two others (20%). Two cases (20%) were given the wrong diagnosis, one of which (plasmacytoma) was not represented previously in MU's memory.

Mammographic Cases Data Analysis

Diagnosis	number in database	p_i	n_i	E_i
carcinoma	5	0.238	3	0.714
ca-inf-ductal	4	0.190	2	0.381
ca-inf-lobular	1	0.048	0	0
ca-colloid	1	0.048	0	0
plasmacytoma	0	0	1	0
met-adeno-in-ln	1	0.048	0	0
met-melanoma	1	0.048	0	0
papilloma	1	0.048	0	0
cyst	2	0.095	0	0
fibroadenoma	5	0.238	4	0.952
Total	21	1.000	10	2.048

Expected number of correct answers: 2.048
Actual number of correct answers: 6
Expected number of incorrect answers: 7.952
Actual number of incorrect answers: 4
χ^2 value (with Yate's correction): 7.3 ($p < 0.01$)

Figure 11: Tables for computing χ^2 value for mammographic cases. The symbols are described in Figure 8. The total number of completely correct guesses for diagnosis was 6, compared to an expected number of 2.048 ($p < 0.01$). If the partially correct guesses are included, for a total of 8 correct or partially correct guesses, we find $\chi^2 = 18.3$ and $p < 0.001$.

4 Discussion

4.1 The Mechanics of MU

Some computer programs for diagnosis function like an inscrutable black box, with a case entered in one side and the diagnosis delivered out the other. Our system, MU, is built on an entirely different model. A human expert reacts to the description of a patient with a wide variety of responses: recollections of similar cases, suppositions about what features ought to be present, puzzlement over unexpected features of the case, judgments on whether the signs and symptoms evoke a particular diagnosis or are merely non-specific. By combining these strategies and chaining together recollections, a human sometimes comes to a serendipitous discovery. We refer to this flexible and heterogeneous mode of thinking as *musing* [28,29]. Our system was designed to provide the varied sorts of commentary that a human might in examining each new case. The musing design together with a case-based memory form the foundation of the cognitive model for MU.

In order to understand how MU works, we must delve into how the data is represented internally. The elemental object that MU manipulates, called a *point* or a *case*, corresponds to an isolated recollection. A *point* can contain a remembrance of a specific patient, a conglomeration of several patients so similar that they have merged together in memory, a generic description of a disease state, a helpful associational rule, or other useful data. Internally, a point is a collection of several features, each one consisting of an *attribute* or

```
((name                    john-doe)
 (age                     56)
 (sex                     male)
 (weight-in-kg            80)
 (recent-weight-loss      no)
 (children's-names        ken rhonda mihaly)
 (favorite-food           chocolate-ice-cream)
 (lives-in                new-london connecticut)
 (hobbies                 camping)
 (smoking-hx-pack-years 50)
 (occupation              shipyard-worker)
 (toxin-exposure          asbestos formaldehyde)
 (cough                   morning non-productive)
 (hemoptysis              no)
 (chest-x-ray             normal)
 (tuberculosis-hx         no)
 (clubbing                no)
 (dyspnea                 occasional)
 (electrocardiogram       normal)
 (hoarseness              no)
 (fever                   yes)
 (fatigue                 yes)
 (headache                yes)
 (rash                    yes)
 (dysuria                 no))
```

Figure 12: A sample point τ used by MU.

dimension along with the *value* of that attribute, as shown in the example of Figure 12. Note that a particular attribute, such as **toxin-exposure**, can possess multiple values simultaneously, here both **asbestos** and **formaldehyde**. Also, there is no limitation on the type of features that are allowed to compose a point; a case has no predetermined slots. This flexibility can be very important, since seemingly irrelevant features such as **(lives-in new-london connecticut)** and **(hobbies camping)** may be the trigger that leads to an unusual diagnosis like Lyme Disease.

Mathematically, each case can be referred to as an unordered n-tuple τ. The system's memory, or database, is simply a set $\mathcal{M}$ of such tuples τ_i. One can think of a feature space spanning all the possible attributes in the system; in such a geometric model, cases are points in the feature space. If diagnoses are well-defined and if the attributes are relevant, we would expect the points to cluster into small groups, each with a single diagnosis. Postulating a diagnosis for a new presenting case reduces to locating the closest cluster to a point in the n-dimensional feature space. Although this is reminiscent of the work of Kulikowski mentioned previously [25], there are several important differences. First, MU allows a case to have missing attributes, as well as allowing an arbitrary number of differing attributes. Our system is strongly domain-independent, and indeed, easily handles cases in several domains simultaneously. The memory $\mathcal{M}$ may contain a heterogeneous mix of points τ_i, say a potpourri of patient cases, mammographic findings, recollections of restaurants, and descrip-

tions of folkdances. These points would likely form into four distinct clusters, perhaps even separating into mutually orthogonal subspaces, but would still be handled in the same manner by MU. In addition, Kulikowski's system handles distance metrics in a strictly mathematical fashion and uses a linear transformation to improve the separation into clusters, while MU uses symbolic techniques of AI in its metrics. Finally, MU uses the metrics merely as a launching point to jump into the more varied commentary of "musing" discussed above.

Having a data structure consisting of a memory of cases provides an obvious method for building expertise into the system: MU simply adds new patient cases incrementally to its database. Automated entry of data is already a key part of several systems [19,20,21]. MU differs from these systems in that it also has a mechanism for stocking the system with prototypical cases before any real patients have been added. One such tuple τ is shown in Figure 13, representing an amalgamation of 10 theoretical cases of pulmonary embolism. The numbers following each value represents the frequency of occurrence. Thus, out of the 10 cases lumped together, 8 were found to have dyspnea (shortness of breath). Prototypical cases such as these serve to "prime the pump," acting as a foundation on which to build. As MU builds a larger and larger database, the new points of patients with pulmonary embolism will outnumber the stocking case τ_0, rendering it insignificant. This is analogous to the process of physician in training, gradually replacing stereotypical right-out-of-the-book cases with actual clinical experiences.

```
((diagnosis              pulmonary-embolism)
 (num-cases              10)
 (dyspnea                (yes 8) (no 2))
 (chest-pain             (pleuritic 4) (non-pleuritic 2)
                         (none 4))
 (chest-x-ray            (normal 7) (infiltrate 1)
                         (effusion 1) (wedge-defect 1))
 (activity               (bed-ridden 5) (post-op 3) (normal 2))
 (onset                  (sudden 9) (gradual 1))
 (hemoptysis             (yes 1) (no 9))
 (calf-pain              (yes 4) (no 6))
 (calf-swelling          (unilateral 3) (bilateral 1) (none 6))
 (v/q-scan               (hi-prob 3) (intermediate-prob 5)
                         (low-prob 2))
 (pulmonary-angiogram 3 (positive 9) (negative 1)))
```

Figure 13: A "prototypical" case for stocking MU, demonstrating both multiple cases condensed into one case and feature-multiplicity. (The numbers given here were chosen for illustrative purposes only and are not clinically accurate.)

The sample case in Figure 13 demonstrates another optional argument that MU understands, called *multiplicity*. The number 3 found in the **pulmonary angiogram** feature indicates that the results of that test are to be considered three times as important as any other feature listed. This provides a way for the user to emphasize that a given feature is especially significant.

Within this framework of $\mathcal{M}$ and τ, one of the primitive operations that MU performs is

$$FETCH(\tau_0, \mathcal{M}) \longrightarrow \mathcal{L},$$

which ranks the points τ_i in $\mathcal{M}$ according to how similar they are to τ_0, a presenting new case, and returns those points in an ordered *remindings list* $\mathcal{L}$. This is equivalent to asking MU, "What other cases does this new case τ_0 bring to mind, in order?"

Of course, this concept of closeness or similarity will depend heavily on the particular distance routines we choose for comparing τ_0 to each case τ_i in $\mathcal{M}$. MU handles this flexibly, allowing the user to define new metrics for specialized attributes. For example, a user may wish to define his own "color-distance," in which **red** is closer to **orange** than it is to **blue**. The distance routines are feature-driven, with each attribute calling up its own metric. If no individualized metric has been specified by the user, MU applies a default metric depending on the type of data. For example, the default distance between two numbers x

and y is given (on a scale of 0 to 100) by:

$$\mathrm{d}(x,y) = 100\,\frac{|x-y|}{\max\{x,y,|x-y|\}}.$$

The overall distance between τ_0 and τ_i is an average of the distances along the features they have in common, weighted by the multiplicity (relative importance of each feature) described above.

Most of the points in $\mathcal{L}$ will lie relatively far away from τ_0. In order to focus attention on the most important points, MU searches for a natural break in the ordered list $\mathcal{L}$ that separates a "close" group of cases $\mathcal{C}$ from a "distant" group of cases $\mathcal{D}$. Thus, MU concentrates on the most relevant memories and ignores those that seem to have little bearing. The details of *CLUSTER*, the algorithm to find this natural division, are described in Appendix B.

Once MU has identified these significant memories $\mathcal{C}$, it proceeds to invoke another primitive operation, *GENERALIZE*, which blends together all the individual cases in $\mathcal{C}$ into a condensed case $\mathcal{Q}$:

$$\mathit{GENERALIZE}(\mathcal{C}) \longrightarrow \mathcal{Q}$$

Is there any distinctive flavor to the resulting stew? Did the presenting case τ_0 bring to mind consistent memories? In order to evaluate this, MU calculates the *evocativeness* of $\mathcal{Q}$, a measure of how successful τ_0 was in stirring up memories with a common diagnosis. If $\mathcal{Q}$ contains twenty cases, nineteen with the diagnosis of **myocardial-infarction** and only one with the diagnosis of **esophageal-reflux**, then τ_0 is deemed highly evocative and significant. On

the other hand, if Q encompasses a dozen equally likely diagnoses, then the features in τ_0 haven't been very helpful. The routine *EVOCATIVENESS* uses an algorithm based on information theory; details are given in Appendix C. This concept of evocativeness is somewhat different from the "evoking strength" used by INTERNIST [14]. MU's evocativeness refers to the strength of τ_0 in focusing attention, and is independent of the specific diagnoses that happen to come to mind. Further, *EVOCATIVENESS* evaluates the significance of an entire presenting case τ_0, not just a single sign or symptom. MU handles the simpler case of calculating the evocativeness of a single feature as well, merely by constructing a case τ consisting of only that single feature. In fact, MU ordinarily does just that, computing the evocativeness individually for each feature in a case and scaling up the *multiplicity* for that feature accordingly. By this method, MU dynamically weights each feature in τ_0 according to its importance.

If the presenting case is partially incomplete, MU attempts to flesh out the missing features with some logical assumptions. The routine *CONCLUDE* determines if any feature-value pair is overwhelmingly present in the cases that are brought to mind. This new information, arrived at from examining the presenting case in the context of the "experience" of the database, may shed some new light on the final diagnosis. Consider, the example shown if Figure 14, with the user asking about patient `barney-rubble` who has findings suggestive of pulmonary embolism. After examining nearby cases, there is a very strong suggestion that the `onset` should be `sudden` and the `pulmonary-angiogram` should

be **positive**; the system temporarily concludes that these features are true.

What new recollections are stirred up by this added information? MU cycles back and reexamines its memory, now finding that it can draw *another* more important conclusion: the **diagnosis** is **pulmonary-embolism**. These cycles of examining the database and drawing inferences allow the system to sift through the database and extract as much information as possible. To some extent, the chaining of conclusions and new recollections corresponds to a cognitive model of free association, as MU focuses its attention down those interesting side trails of reasoning.

While the system tries to conclude as much as can be supported by the presenting case, it carefully avoids contradicting data supplied by the user. Instead, MU draws attention to the situation by *expressing surprise*. The sample output of Figure 14 shows provides one example. Given the initial data, the system would ordinarily have concluded that the value for **calf-swelling** should be **n** ("no"). Since this directly contradicts the user, however, MU declines to draw that conclusion, merely flagging the unusual situation. Similarly, messages are printed when the system draws two contradictory conclusions or when the user contradicts himself. In all cases the information supplied by the user supersedes any assumptions made by the system. These alerting messages serve to provide checks for internal consistency as the user enters a case.

After finishing with the commentary, MU stands back and reevaluates the original goal, coming up with a plausible diagnosis. This is the function of

```
>(new
'( (name barney-rubble) (age 39) (calf-pain y) (calf-swelling n) (homan's-sign y)
(fever n) (white-blood-cell-count normal) (hemoptysis y) (dyspnea y)
(chest-pain pleuritic) (activity bed-ridden) (v/q-scan high-prob) ))

Evocativeness: 4.11

I am reminded of the following cases:

Case Number     NAME                    DIAGNOSIS
-----------     --------------------    --------------------
7               FRED-FLINTSTONE         PULMONARY-EMBOLISM
10              GEORGE-JETSON           PULMONARY-EMBOLISM
4               PEEWEE-HERMAN           PULMONARY-EMBOLISM
16              JANE-SMITH              PULMONARY-EMBOLISM
12              BUGS-BUNNY
21              JOHN-DOE                PULMONARY-EMBOLISM
6               YOGI-BEAR               PULMONARY-EMBOLISM
20              THE-FLASH               CLAUDICATION


Conclusions:
  ONSET                         (SUDDEN)
  PULMONARY-ANGIOGRAM           (POSITIVE)

Evocativeness: 5.0

I am reminded of the following cases:

Case Number     NAME                    DIAGNOSIS
-----------     --------------------    --------------------
4               PEEWEE-HERMAN           PULMONARY-EMBOLISM
6               YOGI-BEAR               PULMONARY-EMBOLISM
7               FRED-FLINTSTONE         PULMONARY-EMBOLISM
10              GEORGE-JETSON           PULMONARY-EMBOLISM
12              BUGS-BUNNY
16              JANE-SMITH              PULMONARY-EMBOLISM
21              JOHN-DOE                PULMONARY-EMBOLISM


:
```

Figure 14: A sample of MU's response to a new case (output continues on the next page). The case is entered at the top, and MU responds with a heterogeneous mix of the evocativeness of the case, cases it is reminded of, surprising features in the case, conclusions, and a guess at the diagnosis.

```
⋮

I'm surprised.
I would have predicted that the CALF-SWELLING would be (Y).


Conclusions:
  DIAGNOSIS                          (PULMONARY-EMBOLISM)

Evocativeness: 5.0

I am reminded of the following cases:

Case Number     NAME                    DIAGNOSIS
-----------     --------------------    --------------------
4               PEEWEE-HERMAN           PULMONARY-EMBOLISM
6               YOGI-BEAR               PULMONARY-EMBOLISM
7               FRED-FLINTSTONE         PULMONARY-EMBOLISM
10              GEORGE-JETSON           PULMONARY-EMBOLISM
16              JANE-SMITH              PULMONARY-EMBOLISM
21              JOHN-DOE                PULMONARY-EMBOLISM

My best guess for DIAGNOSIS is PULMONARY-EMBOLISM

Adding generalization:
(NAME (BARNEY-RUBBLE, FRED-FLINTSTONE))

Forgetting:
(NAME (FRED-FLINTSTONE))
```

Figure 14 (continued).

the routine *GUESS*. The system performs a *FETCH* operation based on its current understanding of the case, which includes the features typed in by the user, the weighting of the features calculated from the individual evocativeness values, and the system's own conclusions. From the cases that come to mind it attempts to find the most likely diagnosis. An example is shown at the bottom of Figure 14, where the system correctly guesses the diagnosis of `pulmonary-embolism`. The function *GUESS* is not limited to speculating on the diagnosis; the user can ask MU about any feature. In evaluating the psychiatric cases, for example, the primary goal was to find the response to desipramine, with a secondary goal of guessing the clinical diagnosis of the patient. We could have just as easily asked the system to guess the age of the presenting patient, based on similar patients it has seen.

MU offers more than just the commentary and guesses described above; it also provides explanatory capabilities. When the user does not understand what prompted a particular recollection, he can use the routine *why*, as shown in Figure 15. MU will summarize the attributes the two cases share, indicating why that case was brought to mind. Lists are provided of the features that are identical, similar and different. By highlighting the attributes they have in common, MU can shed some light on why two seemingly distinct cases are in fact quite similar. If more specific information is needed, the user has access to all the lower-level distance metrics as well.

Another explanatory mechanism is the *DESCRIBE* routine. This function

```
>(new
'( (name barney-rubble) (age 39) (calf-pain y) (calf-swelling n) (homan's-sign y)
(fever n) (white-blood-cell-count normal) (hemoptysis y) (dyspnea y)
(chest-pain pleuritic) (activity bed-ridden) (v/q-scan high-prob) ))

Evocativeness: 4.11

I am reminded of the following cases:

Case Number    NAME                  DIAGNOSIS
-----------    --------------------  --------------------
7              FRED-FLINTSTONE       PULMONARY-EMBOLISM
10             GEORGE-JETSON         PULMONARY-EMBOLISM
4              PEEWEE-HERMAN         PULMONARY-EMBOLISM

:
:
:
>(why 'george-jetson)


Their  DIAGNOSIS, ONSET, PULMONARY-ANGIOGRAM,
       ACTIVITY, CALF-PAIN, DYSPNEA, HEMOPTYSIS,
       V/Q-SCAN  are identical.
Their  AGE  is similar.
Their  CALF-SWELLING, NAME  are different.
```

Figure 15: Explanatory capability using the *WHY* function.

```
> (describe 'diagnosis 'pulmonary-embolism)

The ACTIVITY is BED-RIDDEN
The AGE ranges FROM 21 to 80
The ARTERIAL-PO2 is LOW
The CALF-PAIN is Y
The CALF-SWELLING is Y
The DIAGNOSIS is PULMONARY EMBOLISM
The DYSPNEA is Y
The HEMOPTYSIS is Y
The NAME is PEEWEE-HERMAN, YOGI-BEAR, FRED-FLINTSTONE,
            GEORGE-JETSON, JANE-SMITH, JOHN-DOE
The ONSET is SUDDEN
The PULMONARY-ANGIOGRAM is POSITIVE
The V/Q-SCAN is HIGH-PROB
The CHEST-PAIN is PLEURITIC
The FEVER is Y, N
The HOMAN'S-SIGN is Y
The WHITE-BLOOD-CELL-COUNT is NORMAL
```

Figure 16: The function *DESCRIBE*. To create the disease concept, MU generalizes over all the cases that have the diagnosis pulmonary embolism.

uses *GENERALIZE* to construct an internal representation for a diagnosis in its entirety. To build up a conceptual model of pulmonary embolism, for instance, the system simply culls all the cases of pulmonary embolism from memory and uses *GENERALIZE* to reduce the set to one lumped case. The resulting object reflects the system's current understanding of the disease, as illustrated in Figure 16. This global disease concept corresponds to a frame in a disease-centered database such as those in PIP [11] or INTERNIST [14]. Unlike fixed, static frames, however, the disease concepts in MU are dynamically modified each time a new case is incrementally added to the database. Thus the representation of a dis-

ease evolves as the patient population changes. In earlier versions of MU, the process of guessing relied on these global disease concepts. Instead of using the cases in $\mathcal{M}$ for comparison, the system would compare the presenting case to each one of the disease concepts, to see which one it most resembles. Although this method works well with internally consistent databases, it tends to be less successful when the cases for each diagnosis are heterogeneous.

A user can choose one of several options when asking the system about a new case. The function *QUERY* elicits the sort of running commentary shown in the examples cited previously, but does not add the new case to the database. Another routine, *NEW*, provides the same commentary and also stores the new case in memory. Instead of adding a batch of features all at once, the user can use the routine *INPUT-FEATURES*, entering the attributes of a case one-by-one, and benefiting from the continual commentary and feedback from the system after each feature. Finally, using the routine *APPEND-FEATURES*, the user can incrementally add new attributes to a case previously entered in batch mode.

When adding a case to memory, MU does not always deposit the point unchanged. If the case strongly resembles other cases already stored, the system merges the similar cases together and stores the combined case. We call this mechanism *forgetting*, based on the cognitive model of an expert who can remember several very similar cases but is unable to sort out exactly which details belong to which case. When recalling cases of infiltrating ductal carcinomas of

the breast, for instance, it makes little difference whether Mrs. Smith was 58 years old at diagnosis and Mrs. Jones 68, or vice-versa. The information is still retained, stored in a joint form, but the division between the two cases is blurred. At the end of the output of Figure 14, the presenting case was found to be close enough to one of the cases in memory that the two cases were blended together and the combined case was stored in memory. By using the forgetting mechanism to blend together details, the system achieves saving in storage space.

4.2 Analysis of Testing

The results of testing with the psychiatric cases were summarized previously Figures 6–8, and the results of testing with mammographic cases in Figures 9–11. The system made correct guesses in 50% to 80% of the features, depending on the specific attribute guessed and the strictness of the definition of "correct." The system did not do well in guessing the response of the psychiatric patients to desipramine, but performed significantly well when guessing the clinical diagnosis of the psychiatric patients and the pathologic diagnosis of the mammography cases.

There are several plausible explanations for less than perfect performance. The size of the databases were relatively small; with only 32 psychiatric cases and 20 mammographic cases to stock the system. This makes the system dependent on the quirks of particular cases that may not be smoothed out by other cases in the database with the same diagnosis. Further, when the system was

trying to guess at a rare attribute, the small size of the database became particularly important. In guessing the diagnosis of mammographic case number 30, for example, MU was severely handicapped from the start, having never seen a case of plasmacytoma (an uncommon tumor of the breast). The closest diagnosis it could muster up was fibroadenoma. In essence MU was still in its medical training, and had not seen enough "zebras" to arrive at an unusual diagnosis. Another possible source of error could arise from a lack of internal consistency among the cases. For instance, unless the patients responding to desipramine are indeed linked together by some similar attributes, any consultant would be hard-pressed to predict drug response.

While counting the number of correct guesses that MU makes provides a simple standard for gauging accuracy, it ought not be the only measure of MU's success. Guessing the diagnosis is only one aspect of the multi-faceted running commentary from the system. Indeed, information provided by unusual cases in the remindings list or surprises in the presenting case may prove more valuable in suggesting a final diagnosis to the user. For MU, the process of musing is as important as the final goal.

For example, in the mammography case of Figure 9, shown earlier, the system finds that the case in memory that most closely matches the presenting case carries a diagnosis of colloid carcinoma. The majority of the cases that it recollects, however, are of a benign nature and several are fibroadenomas (FA). Making its judgement in the context of the entire database, MU guesses that

the presenting case, too, is a fibroadenoma. The system alerts the user, though, that one of the attributes of the case appears unusual: from the cases brought to mind, it would seem that a "comet" should not be present radiographically. A comet can be a sign of malignancy and is not typical of the benign tumor fibroadenoma. MU handles the situation smoothly, making a tentative diagnosis while still informing the user of atypical features. (As icing on the cake, MU made the correct diagnosis in this case.)

4.3 Comments

Using actual patient cases in the database has both advantages and disadvantages. Although the system performed fairly well when guessing, the limitations of having a small database were apparent. Some diagnoses, such as plasmacytoma, simply were not represented in the system's memory; others, such as colloid carcinoma, were found only in single cases. When diagnoses are infrequently found in the database, the peculiarities of each case take on an exaggerated role. The system has no choice but to construct a disease concept based on an isolated case report. Some of these problems can be avoided by using templates to stock the system, allowing the addition of new cases to gradually overcome and replace the templates in importance.

Of course, the commentary that MU provides will be highly dependent on the mix of cases that are entered. The system is essentially a "local doctor," familiar with the diseases that are common to patients in the area, changing the order in a differential diagnosis list as the patient population of the area slowly

changes. An implementation of MU in Connecticut, for example, would likely contain descriptions of patients with Lyme disease, while an implementation in another state might not. Once stocked with a large number of cases, the system works quite efficiently on diagnosing the locally frequent diseases, but the database loses some portability in the process.

Another disadvantage of our case-based expert system is that there is no simple representation in the present system for complex interrelationships between items of data. Most of the links in the database will be associational links, with no natural method for storing other types of knowledge such as cause and effect relationships or anatomical knowledge. One way to circumvent this problem is simply to add more flexible features to each case. We can store *procedural* knowledge right alongside our factual knowledge. Each case might possess a feature, for instance, that spurs MU into investigating a differential diagnosis list, asking about laboratory test results, and toward searching for an etiology. The resulting data-points combine the advantages of case-based systems and frame-based systems. Temporal information can easily be incorporated by providing a feature that marks the time and day that a new case was entered.[1]

Despite the high degree of flexibility already built into the database, which can accommodate arbitrary features, there must be some standardization of cases for the system to be able to form its associational chains. It might be difficult for the system to recognize that `serum-sodium`, `serum-Na` and `Na` all

[1] Just such a "time-stamp" has been implemented by Scott Fertig in the Computer Science Department at Yale, in a successor system to MU.

referred to the same attribute. Various solutions come to mind. One person could be responsible for entering all the cases in the system and maintaining internal consistency. This is basically the path chosen in the system's development to date. Alternatively, a menu-driven interface can assist an inexperienced user in entering a case [28]. The user would be presented with a list of possible features, and can enter a partial feature (such as `sodium`) to prompt the system for the preferred usage.

One advantage of a case-based system over a rule-based system is that the case-based system can deal with internal paradoxes. Two cases with exactly the same features save differing diagnoses can peacefully coexist. There is no need to satisfy formal logical consistency. Similarly, a highly unusual case that "breaks all the rules" needs no special handling—whereas a rule-based system would have to build meta-rules, rules that govern the application of other rules, to handle the inconsistencies.

Another advantage is that MU has a simple mechanism for meta-knowledge: knowing what it knows. It is easy to determine how many memories are stirred up by a presenting case and to check if they are alike or quite different, so MU can recognize how much confidence it ought to have in its information, and where its limitations are. When too few memories are recollected or when the remembered cases are too distant from the presenting case, the system responds by stating "That doesn't remind me of anything in particular," and forgoes any further analysis based on shaky data.

Because the system's understanding of a case does not depend on locating appropriate rules or frames to be applied to the case, MU can better manipulate novel information. MU can conceive of objects that it has never seen before. For example, a user can query MU about what a "purple cow" might be like, though MU has never seen a purple cow. It is difficult to imagine how a rule-based or a frame-based system would deal with this unusual object. MU, on the other hand, merely proceeds with its standard fetch operation, recollecting memories associated with the color and the animal. Even with no previous experience of seeing a purple cow, the system can come to a reasonable conclusion. If all the cases of purple objects in memory are flowers and berries with strong sweet odors, and all the cows in memory have strong objectionable odors, MU could reasonably suppose that a purple cow has a strong odor of some type. The concept that MU forms of "purple cow" comes from generalizing over the group of somewhat dissimilar memories that are called to mind. Thus, the flexibility of musing allows the system to form new schemas for concepts that do not fit into established rules or frames.

4.4 Summary

In summary, the medical expert system MU relies on a cognitive model of a case-based memory and a style of reasoning termed *musing*, involving associational chaining and a varied and flexible commentary. Preliminary testing of the system demonstrated reasonable performance in several domains; however, larger databases are needed for full-scale trials of the system's capabilities.

Appendix A: Code

This appendix contains the full computer code for the program MU, written in the language T. It consists of approximately 1500 lines divided among some 140 procedures.

```
;;;================================================================

(herald mu)


;;;
;;; LOAD MACROS:
;;;   "FOR" macro and "=>" macro used in code
;;;   "TRACE" macro used in debugging
;;;

(load "/homes/systems/dhg/mu/frozen/for.mobj")
(load "/homes/systems/dhg/mu/frozen/msg.mobj")
(load "/homes/systems/dhg/mu/frozen/sf:macros.mobj")
(load "/homes/systems/dhg/mu/frozen/sf:trace.mobj")

;;;
;;; STRUCTURE DEFINITIONS
;;;

;;; define new structure type for a remindings list entry
;;;   with fields shown below, and printing instructions
(define-structure-type entry  name
                              distance
                              raw-score
                              dims-in-common
                              num-axes
                              num-cases
                              index
  (((print self port)
    (format port
            "(~s ~s ~s ~s ~s ~s ~s)"
            (entry-name self)
            (round (entry-distance self) 1)
            (round (entry-raw-score self) 1)
            (entry-dims-in-common self)
            (entry-num-axes self)
            (entry-num-cases self)
            (entry-index self)
    ))))

;;; makes a dummy remindings list entry will all nil features
```

```
(define (make-nil-entry)
  (let ((nil-entry (make-entry)))
    (set (entry-name      nil-entry)       'nil-entry)
    (set (entry-distance  nil-entry)       101)
    (set (entry-raw-score nil-entry)       101)
    (set (entry-dims-in-common nil-entry)  0)
    (set (entry-num-axes  nil-entry)       0)
    (set (entry-num-cases nil-entry)       0)
    (set (entry-index     nil-entry)      -1)
    nil-entry))

;;;
;;;  CONSTANTS
;;;

; minimum distance needed for two cases to be considered contradictory
(define-constant *contradict-threshold* 10)
; minimum number of cases needed from which to draw conclusions
(define-constant *case-threshold* 2)
; minimum fraction of findings which must be consisten to draw a conclusion
(define-constant *fraction* .85)
; marker for if-then rules
(define-constant *marker* 1000)
; maximum distanct between cases that can be blended together
(define-constant *forget-threshold* 10)
; minimum number of attributes in common before cases can be blended together
(define-constant *forget-fraction* .74)
; global minimum for evocativeness value
(define-constant *min-evoc* 0.5)
; global maximum for evocativeness value
(define-constant *max-evoc* 5)

;;;
;;; GLOBALS
;;;

;;; initialize global variables
(lset *mem* '() )
(lset *remindings* '() )
(lset *current* '() )
(lset *conclusions* '())
(lset *max-entropy* 1)
(lset *nil-entry* (make-nil-entry))

(lset *named-dim* 'name)
(lset *special-dim* 'name)
(lset *batch-mode* NIL)

(lset *new-one* NIL)
(lset *metric-table* (make-table))

;;;
;;; printing routines
;;;
```

```
(define (print-r-info r-lst top-cluster)
    (format T "~%~%Remindings list:~%")
    (newline (standard-output))
    (multiple-entry-print r-lst (standard-output))
    (format T "~%~%Top cluster of remindings:~%")
    (newline (standard-output))
    (multiple-entry-print top-cluster (standard-output))
    (newline (standard-output)))

(define (val-lst->string val-lst)
    (cond (  (null? val-lst)
             ""  )
          (  (every? (lambda (elt) (number? (car elt))) val-lst)
             (collapse val-lst)  )
          (  (null? (cdr val-lst))
             (symbol->string (caar val-lst))  )
          (  else
             (string-append (symbol->string (caar val-lst))
                            ", "
                            (val-lst->string (cdr val-lst)))  )))


;;; Prints three objects (as best as possible) in three columns,
;;;  here set at columns 0, 15, 40
;;; Used by recall-names.
(define (print-in-three-columns r-lst)
    (format (standard-output) "~a" "Case Number")
    (set (hpos (standard-output)) 15)
    (format (standard-output) "~a" *named-dim*)
    (set (hpos (standard-output)) 40)
    (format (standard-output) "~a~%" *special-dim*)
    (format (standard-output) "~a" "-----------")
    (set (hpos (standard-output)) 15)
    (format (standard-output) "~a" "--------------------")
    (set (hpos (standard-output)) 40)
    (format (standard-output) "~a~%" "--------------------")
    (walk (lambda (entry)
              (format (standard-output) "~a" (entry-index entry))
              (set (hpos (standard-output)) 15)
              (format (standard-output) "~a"
                      (val-lst->string (entry-name entry)))
              (set (hpos (standard-output)) 40)
              (format (standard-output) "~a~%"
                  (val-lst->string
                      (val-lst-of (nth *mem* (entry-index entry))
                                  *special-dim*)))
              NIL)
          r-lst)
    (newline (standard-output)))

(define (print-in-two-columns r-lst)
    (format (standard-output) "~a" "Case Number")
    (set (hpos (standard-output)) 15)
    (format (standard-output) "~a~%" *named-dim*)
    (format (standard-output) "~a" "-----------")
```

```
    (set (hpos (standard-output)) 15)
    (format (standard-output) "~a~%" "--------------------")
    (walk (lambda (entry)
              (format (standard-output) "~a" (entry-index entry))
              (set (hpos (standard-output)) 15)
              (format (standard-output) "~a~%"
                      (val-lst->string (entry-name entry)))
              NIL)
          r-lst)
    (newline (standard-output)))

;;; Prints out a little chart with cases from a remindings list.
;;; Includes in its info:
;;;     case number
;;;     name of the case  (i.e. its value along *named-dim*)
;;;     diagnosis of case (i.e. its value along *special-dim*)
(define (recall-names top-cluster)
    (cond (  (null? top-cluster)
             (newline (standard-output))
             (writes (standard-output)
               "That doesn't remind me of anything in particular.")
             (newline (standard-output))
             (newline (standard-output))  )
          (  else
             (newline (standard-output))
             (writes (standard-output)
                 "I am reminded of the following cases:")
             (newline (standard-output))
             (newline (standard-output))
             (if (or (eq? *named-dim* *special-dim*)
                     (eq? *special-dim* NIL))
                 (print-in-two-columns top-cluster)
                 (print-in-three-columns top-cluster))  )))

;;;
;;; ADDING NEW CASES TO MEMORY
;;;

(define (addelt new-case)
    (install new-case)
    (let* (  (r-lst     (cluster (fetch-r-lst new-case)))
             (name      (name-of new-case))
             (num       (num-cases-of new-case))
             (index-lst (del = 0 (forget-indices r-lst new-case num)))  )
          (cond (  index-lst
                   (forget-if-necessary index-lst)
                   repl-wont-print)
                (  name
                   (format T "~%Adding new case:~%")
                   (print name (standard-output))
                   (newline (standard-output)))
                (  else T  ))))

(define (forget-indices r-lst new-case num)
    (cond (  (null? r-lst) NIL  )
```

```
          (  (> (entry-distance (car r-lst)) *forget-threshold*)
             NIL  )
          (  (and (not (zero? (entry-num-cases (car r-lst))))
                  (not (zero? num))
                  (> (entry-dims-in-common (car r-lst))
                     (* *forget-fraction* (entry-num-axes (car r-lst))
;                       (max (entry-num-axes (car r-lst)) num)
                      ) ) )
             (cons (entry-index (car r-lst))
                   (forget-indices (cdr r-lst) new-case num))  )
          (  else
             (forget-indices (cdr r-lst) new-case num)  )))

;;; If entering this function, then will be adding at least one generalization
;;; to the database (and deleting more specific points).
(define (forget-if-necessary index-lst)
    (let ( (gen (generalize (cons (car *mem*)
                                  (map (lambda (index) (nth *mem* index))
                                       index-lst)))) )
      (format T "~%Adding generalization:~%")
      (pretty-print gen (standard-output))
      (newline (standard-output))
      (newline (standard-output))
      (set (car *mem*) gen) )
    (map (lambda (index)
            (format T "~%Forgetting:~%")
            (pretty-print (nth *mem* index) (standard-output))
            (newline (standard-output))
            (newline (standard-output))
            (set (nth *mem* index) NIL))
         index-lst)
    (set *mem* (delq '() *mem*)))


(define (install new-case)
    (set *mem* (cons new-case *mem*))
    (set *max-entropy*
         (cond (  (val-lst-of (generalize *mem*) *special-dim*)
                  =>
                  (lambda (x) (log (->float (length x)))))
               (  else -50  ))))


;;;
;;; BASIC ACCESS ROUTINES FOR USER
;;;

;;; ask about a case but do not add it to memory
(define (query lst)
    (let (  (temp (standardize lst))  )
         (set *new-one* temp)
         (set *current* NIL)
         (set *conclusions* NIL)
         (set *remindings* NIL)
         (process temp NIL)
```

```
        T))

;;; ask about a case and when finished, add to memory
(define (new lst)
    (query lst)
    (addelt *new-one*))

;;; input features one-by-one
(define (input-features)
    (set *new-one* NIL)
    (set *current* NIL)
    (set *conclusions* NIL)
    (set *remindings* NIL)
    (append-features)
    (addelt *new-one*))

;;; append features one-by-one to old case
(define (append-features)
    (do (  (feature (get-feature) (get-feature))
           (fact          NIL           NIL     )  )
        (  (eof? feature) 'all-done  )
        (set fact (list (standardize-feature feature 1)))
        (cond (  (eq? (caar fact) 'help)
                 (help (caadar fact))  )
              (  else
                 (process fact NIL)
                 (set *new-one*
                      (blend-cases fact *new-one*)) )))
    (set *new-one* (standardize *new-one*))
    T)

(define (get-feature)
  (format T "-- ")
  (read-object (terminal-input) *standard-read-table*))


;;;
;;; PROCESS ROUTINES:  THE HEART OF MU'S CYCLE
;;;

(define (process new-features conclude-flag)
    (let* (  (kept-features
              (for (feature in new-features)
                   (filter (process-feature feature conclude-flag))))
             (r-lst          (fetch-r-lst kept-features))
             (evoc           (evocativeness r-lst))
             (top-cluster    (cluster r-lst))
             (point          (generalize (r-lst->points top-cluster)))
             (num            (max (* *fraction* (num-cases-of point))
                                  *case-threshold*))  )
          (cond (  (null? kept-features)
                   (guess *special-dim*)  )
                (  else
                   (if *batch-mode*
                       (set kept-features
```

```
                    (map (lambda (feature)
                           (scale-up feature evoc))
                         kept-features)))
              (cond (  conclude-flag
                       (print-conclusions kept-features)
                       (set *conclusions*
                            (blend-cases kept-features
                                         *conclusions*))  )
                    (  else NIL  ))
              (format (standard-output)
                      "~%Evocativeness: ~s~%" (round evoc 2))
              (set *current*
                   (blend-cases kept-features *current*))
              (recall-names top-cluster)
              (process (conclude-point point num) T)  ))))

(define (process-feature feature conclude-flag)
    (let* (  (rescaled-feature
               (if *batch-mode*
                   feature
                   (scale-up feature
                       (evocativeness (fetch-r-lst (list feature))))))
             (old-conclusion      (assq (car feature) *conclusions*))
             (old-user-entry      (assq (car feature) *new-one*))
             (old-current         (assq (car feature) *current*))
             (contradicts-system? (contradicts? feature old-conclusion))
             (contradicts-user?   (contradicts? feature old-user-entry))  )
          (express-surprise feature
                            old-conclusion
                            old-user-entry
                            contradicts-system?
                            contradicts-user?
                            conclude-flag)
          (cond (  (and conclude-flag
                        contradicts-user?)
                   NIL  )
                (  else
                   (cond (  contradicts-system?
                            (delq! old-conclusion *conclusions*)
                            (delq! old-current *current*)  )
                         (  else NIL  ))
                   (cond (  contradicts-user? ;and (not conlcude-flag)
                            (delq! old-user-entry *new-one*)
                            (delq! old-current *current*)  )
                         (  else NIL  ))
                   rescaled-feature  ))))

(define (contradicts? feature1 feature2)
    (cond (  (or (null? feature1) (null? feature2))
             NIL  )
          (  (not (eq? (car feature1) (car feature2)))
             NIL  )
          (  else
             (let* (  (entry1 (compare (list feature1)
                                       (list feature2)))
```

```
                    (distance (entry-distance       entry1))
                    (dims     (entry-dims-in-common entry1))  )
               (and (not (= 0 dims))
                    (> distance *contradict-threshold*)))  )))

(define (scale-up feature evoc)
    (cond (  (eq? (car feature) 'num-cases)
             feature  )
          (  (<= evoc (cadr feature))
             feature  )
          (  else
             (list (car feature) evoc (caddr feature))  )))

(define (express-surprise feature
                          old-conclusion
                          old-user-entry
                          contradicts-system?
                          contradicts-user?
                          conclude-flag)
    (cond (  (and (not conclude-flag)
                  contradicts-user?)    ;user contradicting user
             (format (standard-output)
               "~%I'm surprised.~%You told me before that the ")
             (format (standard-output)
               "~s was ~s.~%~%"
               (car feature)
               (collapse (cddr old-user-entry)))  )
          (  (and conclude-flag
                  contradicts-user?)    ;system contradicting user
             (format (standard-output)
               "~%I'm surprised.  I would have predicted that the ")
             (format (standard-output)
               "~s would be ~s.~%~%"
               (car feature)
               (collapse (cddr feature)))  )
          (  contradicts-system?        ;user or system contradicts
                                        ;system
             (format (standard-output)
               "~%I'm surprised.~%I had assumed before that the ")
             (format (standard-output)
               "~s was ~s.~%~%"
               (car feature)
               (collapse (cddr old-conclusion)))  )))

(define (print-conclusions conclusion-lst)
    (cond (  conclusion-lst
             (format T "~%Conclusions:~%")
             (map (lambda (elt)
                      (format T
                              "  ~s~30T    ~s~%"
                              (car elt)
                              (collapse (cddr elt)))
                      NIL)
                  conclusion-lst)  )
          (  else NIL  )))
```

```
;;;
;;; EVOCATIVENESS:  HOW STRONGLY DOES A CASE EVOKE ONLY A FEW DIAGNOSES?
;;;

(define (evocativeness r-lst)
    (let* (  (top-cluster (cluster r-lst))
             (num-lst     (map cadr (r-lst->val-lst top-cluster
                                                    *special-dim*)))
             (ratio       (scaled-entropy num-lst))  )
          (+ (*    ratio    *min-evoc*)
             (* (- 1 ratio) *max-evoc*))))

;;;  Returns a number 0<=n<=1
;;;  If total is zero, nothing was present for *special-dim*
;;;    and the entropy is 1.
;;;  If *max-entropy* = 0, there is exactly one value in all of
;;;    *mem* for *special-dim*.  If total>0, we must have found
;;;    that value of *special-dim*, and therefore entropy is 0.
;;;  If *max-entropy* < 0, this is a flag that there are no values
;;;    anywhere in *mem* for *special-dim*, and nothing could possibly
;;;    be brought to mind about *special-dim*.  This should have
;;;    been trapped earlier by (=0? total) -- if it was not,
;;;    someone needs to reset *max-entropy* which must be wrong.
;;;  If any values in rep-lst are negative, something is wrong.
;;;  If (log (->float (length rep-lst))) is greater than
;;;    *max-entropy*, there are more values of *special-dim*
;;;    brought to mind than are supposed to exist in all of *mem*.
;;;    In that case, *max-entropy* needs to be reset.
;;;  To reset *max-entropy*,
;;;  (set *max-entropy*
;;;    (log (->float (length
;;;       (val-lst-of (generalize *mem*) *special-dim*)))))
;;;
(define (scaled-entropy rep-lst)
    (let (  (total (sum-of rep-lst))  )
         (cond (  (=0? total) 1  )
               (  (and (= *max-entropy* 0)
                       (>0? total))
                  0  )
               (  (<0? *max-entropy*)
                  (warn 'entropy-function rep-lst *max-entropy*)
                  1  )
               (  (any? <=0? rep-lst)
                  (warn 'entropy-function rep-lst *max-entropy*)
                  1  )
               (  (> (log (->float (length rep-lst)))
                     *max-entropy*)
                  (warn 'entropy-function rep-lst *max-entropy*)
                  1  )
               (  else
                  (/ (entropy-function rep-lst total)
                     *max-entropy*)  ))))
```

```
(define (r-lst->val-lst r-lst dim)
    (cond (  (null? r-lst) NIL  )
          (  else
             (blend-val-lsts (val-lst-of
                                (nth *mem* (entry-index (car r-lst)))
                                dim)
                             (r-lst->val-lst (cdr r-lst) dim))  )))



(define (entropy-function rep-lst total)
    (- (log (->float total))       ;log needs flonum
       (/ (sum-of rep-lst entropy-function1)
          total)))

(define (entropy-function1 number)
    (cond (  (or (= 1 number) (<=0? number))
             0  )
          (  else
             (* number (log (->float number)))  )))  ;log needs flonum

;;;
;;; FETCH:  BASIC MEMORY-EXAMINING ROUTINE
;;;

(define (fetch-r-lst new-case)
    (let (  (index -1)  )
         (map (lambda (point)
                  (increment index)
                  (let (  (temp (compare new-case point))
                          (name (val-lst-of point *named-dim*))  )
                       (set (entry-name temp) name)
                       (set (entry-index temp) index)
                       temp))
              *mem*)))

(define (blend-r-lsts r-lst1 r-lst2)
    (cond (  (null? r-lst1) r-lst2  )
          (  (null? r-lst2) r-lst1  )
          (  else
            (map blend-r-lsts1 r-lst1 r-lst2)  )))

(define (blend-r-lsts1 elt1 elt2)
    (let* (  (entry (copy-structure elt1))  )
          (set (entry-raw-score entry)
               (+ (entry-raw-score elt1)
                  (entry-raw-score elt2)))
          (set (entry-dims-in-common entry)
               (+ (entry-dims-in-common elt1)
                  (entry-dims-in-common elt2)))
          (set (entry-distance entry)
               (mood (entry-raw-score entry)
                     (entry-dims-in-common entry)))
          entry))
```

```
;;;
;;; CONCLUDE
;;;

(define (conclude-point point num)
    (if (null? point)
        NIL
        (let* (  (new-feature (conclude-feature (car point) num))  )
              (if (null? new-feature)
                  (conclude-point (cdr point) num)
                  (cons new-feature
                        (conclude-point (cdr point) num))))))

(define (conclude-feature feature num)
    (if (null? feature)
        NIL
        (let* (  (dim  (car feature)                          )
                 (vals (conclude-val-lst (cddr feature) num dim))  )
              (cond (  (null? vals) NIL  )
                    (  (eq? dim 'num-cases) NIL  )
                    (  else
                       (append (list dim -1) vals)  )))))

(define (conclude-val-lst val-lst num dim)
    (cond (  (null? val-lst) NIL  )
          (  (and (> (cadar val-lst) num)
                  (< (cadar val-lst) *marker*)
                  (not (assq (caar val-lst)
                             (val-lst-of *current* dim))))
             (cons (list (caar val-lst) 1)
                   (conclude-val-lst (cdr val-lst) num dim))  )
          (  else
             (conclude-val-lst (cdr val-lst) num dim)  )))

;;;
;;; COMPARE:  FOR DISTANCE METRICS
;;;

(define (compare newpoint oldpoint)
    (cond (  (or (null? oldpoint) (null? newpoint))
             *nil-entry*  ) ;;; not really what we want?
          (  else
           (let ((new-entry (make-entry)))
             (let* (  (dist-dim       (compare1 newpoint
                                                oldpoint
                                                (list 0 0)
                                                (num-cases-of newpoint)
                                                (num-cases-of oldpoint)))
                      (raw-dist       (car dist-dim))
                      (dims-in-common (cadr dist-dim))
                      (score          (mood raw-dist dims-in-common))  )
                (set (entry-name      new-entry) 'unlabeled)
                (set (entry-distance  new-entry) score )
                (set (entry-raw-score new-entry) raw-dist)
```

```
            (set (entry-dims-in-common new-entry) dims-in-common)
            (set (entry-num-axes new-entry)  (subtract1 (length oldpoint)))
            (set (entry-num-cases new-entry) (num-cases-of oldpoint))
            (set (entry-index new-entry)     -1)
            new-entry ))) ))

(define (compare1 newpoint oldpoint dist-dim num1 num2)
    (cond (  (null? newpoint) dist-dim  )
          (  else
             (compare1 (cdr newpoint)
                       oldpoint
                       (update-dim (car newpoint)
                                   (assq (caar newpoint) oldpoint)
                                         ;should we use val-lst-of here?
                                   dist-dim
                                   num1
                                   num2)
                       num1
                       num2)  )))

(define (mood distance dims-in-common)
    (if (zero? dims-in-common)
        101
        (->float (/ distance  dims-in-common))))

(define (update-dim newfeature oldfeature dist-dim num1 num2)
    (cond (  (null? oldfeature) dist-dim  )
          (  (null? newfeature) dist-dim  )
          (  (eq? (car newfeature) 'num-cases) dist-dim  )
          (  else
             (let (  (multiplicity (max (abs (cadr newfeature))
                                        (abs (cadr oldfeature))))  )
                  (list (+ (car dist-dim)
                           (* multiplicity
                              (distance-between (car newfeature)
                                                (cddr newfeature)
                                                (cddr oldfeature)
                                                num1
                                                num2)))
                        (+ (cadr dist-dim)
                           multiplicity)))  )))

;;;
;;; CLUSTER
;;;

(define (cluster r-lst)
    (let (  (temp (alphabetize r-lst entry-distance))  )
         (cond (  (null? temp)              NIL    )
               (  (>= (entry-distance (car temp)) 100) NIL    )
               (  (and (zero? (entry-distance (car temp)))
                       (>= (entry-dims-in-common (car temp)) 2)
                       (zero? (entry-num-cases (car temp))))
                  (cluster (cdr temp))            )
               (  (null? (cdr temp))       temp  )
```

```
            (  else
               (let* (  (num1      (entry-distance (car temp)))
                        (mod-lst   (common (cdr temp)))
                        (sum-nums  (sum-of mod-lst entry-distance))
                        (left      (length mod-lst)    )  )
                     (reverse (cluster1 (list (car temp))
                                        num1
                                        1
                                        (cdr temp)
                                        sum-nums
                                        left)))  ))))

(define (cluster1 r1 sr1 n1 r2 sr2 n2)
    (if (null? r2)
        r1
        (let* (  (m-r1   (cons (car r2) r1)  )
                 (m-r2   (cdr r2)           )
                 (m-sr1  (+ sr1 (entry-distance (car r2))) )
                 (m-sr2  (- sr2 (entry-distance (car r2))) )
                 (m-n1   (+ n1 1)          )
                 (m-n2   (- n2 1)          )  )
              (cond (  (> (weight sr1 n1 sr2 n2)
                          (weight m-sr1 m-n1 m-sr2 m-n2))
                       r1  )
                    (  else
                       (cluster1 m-r1 m-sr1 m-n1
                                 m-r2 m-sr2 m-n2)  )))))

(define (common r-lst)
    (cond (  (null? r-lst) NIL  )
          (  (> (entry-distance (car r-lst)) 100) NIL  )
          (  else
             (cons (car r-lst) (common (cdr r-lst)))  )))

;;; can call (sum-of '(1 2 3 4 5)) => 15
;;; or can call (sum-of '(1 2 3 4 5) (lambda (x) (* x x))) => 55
(define (sum-of . parameters)
    (cond (  (null? (cdr parameters))
             (sum-of1 (car parameters) (lambda (x) x))  )
          (  else
             (sum-of1 (car parameters) (cadr parameters))  )))


(define (sum-of1 l function)
    (iterate *loop (  (result 0) (lst l)  )
        (cond (  (null? lst) result  )
              (  else
                 (*loop (+ result (function (car lst)))
                        (cdr lst))  ))))

(define (weight sum1 n1 sum2 n2)
    (if (or (zero? n1) (zero? n2))
        0
        (let* (  (mean-diff (- (/ sum1 n1) (/ sum2 n2)))  )
              (* n1 n2 mean-diff mean-diff))))
```

```
(define (delete-unnamed all-names)
   (let* (  (unnamed (memq? '? all-names))
            (names (remove-elt '? (remove-elt '() all-names)))  )
         (list unnamed names)))

;;;
;;; WHY
;;;

(define (why some-name)
     (why-number (index-of some-name)))

(define (why-num index)
    (format T
            "~%Case ~s: (marked entries identical to the current case)~%~%"
            index)
    (walk why-feature (nth *mem* index))
    repl-wont-print
    T)

(define (why-feature feature)
    (cond (  (eq? (car feature) 'num-cases)
             NIL  )
          (  (assq (car feature) *current*)
             (let (  (dist   (car (feature-distance feature
                                        (assq (car feature) *current*))))  )
                  (cond (  (zero? dist)
                           (format T "**")  )
                        (  (< dist 3)
                           (format T "* ")  )
                        (  else
                           (format T "  ")  )))
             (format T "  ~s ~s~%"
                       (car feature)
                       (collapse (cddr feature)))  )
          (  else
             (format T "  ")
             (format T "  ~s ~s~%"
                       (car feature)
                       (collapse (cddr feature)))  )))

(define (feature-distance feature1 feature2)
    (distance-between (car feature1)
                      (cddr feature1)
                      (cddr feature2)
                      1
                      1))

(define (why-number index)
     (let* (  (worklist  (workup index))
              (idents (identicals worklist))
              (fraternals (similars worklist))
              (aliens (strangers worklist))   )
          (cond (  (null? worklist)
```

```
                (format T "I wasn't reminded of that~&")  ))
        (cond (  idents (print-props idents)
                        (format T " identical~&")  ))
        (cond (  fraternals (print-props fraternals)
                         (format T " similar~&")  ))
        (cond (  aliens (print-props aliens)
                         (format T " different~&")  ))
        T
    ))

;;;
;;; WORKUP
;;;

(define (workup index)
    (cond (  (not (number? index))     NIL  )
          (  (> index (length *mem*))  NIL  )
          (  else
             (workup-all *current* (nth *mem* index))  )))

(define (workup-all newpoint oldpoint)
    (cond (  (null? newpoint) NIL  )
          (  (null? oldpoint) NIL  )
          (  else
             (let (  (num1 (num-cases-of newpoint))
                     (num2 (num-cases-of oldpoint))  )
                  (delq '() (map (lambda (feature)
                                         (workup-one feature
                                                     (assq (car feature)
                                                           oldpoint)
                                                     num1
                                                     num2))
                                 newpoint)))  )))

(define (workup-one newprop oldprop num1 num2)
    (cond (  (null? oldprop)                NIL  )
          (  (null? newprop)                NIL  )
          (  (eq? (car newprop) 'num-cases) NIL  )
          (  else
             (list (car newprop)
                   (distance-between (car newprop)
                                     (cddr newprop)
                                     (cddr oldprop)
                                     num1
                                     num2))  )))

(define (identicals workup-list)
  (remove-elt NIL (map  (lambda (elt)
                          (cond  (  (<= (cadr elt) 0)
                                    (car elt)  )
                                 (  else
                                    NIL  ) ))
                workup-list)))

(define (similars workup-list)
```

```
  (remove-elt '()  (map  (lambda (elt)
            (cond  (  (and (< (cadr elt) 100)
                           (> (cadr elt) 0))
                      (car elt)  )
                   (  else '() ) ) )
            workup-list)))

(define (strangers workup-list)
  (remove-elt '()  (map  (lambda (elt)
            (cond  (  (>= (cadr elt) 100)
                      (car elt)  )
                   (  else
                      '() ) ) )
            workup-list)))

(define (print-props prop-list)
   (format T "Their ~s"  (car prop-list))
   (cond (  (null? (cdr prop-list))
            (format T " is ")  )
         (  else (walk  (lambda (elt)
                          (format T ",")
                          (space (standard-output))
                          (format T "~s" elt))
                        (cdr prop-list))
                 (format T " are ")  ))
   repl-wont-print)

;;;
;;; DESCRIBE
;;;

(define (describe dim value)
    (let* (  (instances   (call-to-mind dim value))
             (feature-lst (generalize instances) )  )
          (print-desc feature-lst)))

(define (print-desc summary)
    (let* (  (dim (caar summary))
             (val-lst (cddar summary))  )
          (cond (  (null? dim) T  )
                (  (number? (caar val-lst))
                   (format t "The ~s ranges from ~d to ~d~&"
                           dim
                           (find-min val-lst)
                           (find-max val-lst))
                   (print-desc (cdr summary))  )
                (  else
                  (format T "The ~s is ~s"
                          dim
                          (caar val-lst))
                  (walk (lambda (elt)
                          (format T ", ~s" (car elt)))
                        (cdr val-lst))
                  (newline (standard-output))
                  (print-desc (cdr summary))  ))))
```

```
(define (find-min lst)
   (find-min1 (cdr lst) (caar lst)))

(define (find-min1 lst currmin)
   (cond (  (null? lst) currmin  )
         (  (< (caar lst) currmin) (find-min1 (cdr lst) (caar lst))  )
         (  else (find-min1 (cdr lst) currmin)  )))

(define (find-max lst)
   (find-max1 (cdr lst) (caar lst)))

(define (find-max1 lst currmax)
   (cond (  (null? lst) currmax  )
         (  (> (caar lst) currmax) (find-max1 (cdr lst) (caar lst))  )
         (  else (find-max1 (cdr lst) currmax)  )))


;;;
;;; GUESS
;;;

(define (guess dim)
    (set *remindings* (fetch-r-lst *current*))
    (let* (  (top-cluster    (cluster *remindings*)                  )
             (point          (generalize (r-lst->points top-cluster)))
             (guess-lst      (val-lst-of point dim)                  )  )
          (best-guess guess-lst dim)))

(define (best-guess guess-lst dim)
    (format T "My best guess for ~s is ~s~%"
              dim
              (best-guess1 guess-lst NIL 0))
    T)

(define (best-guess1 guess-lst val max-occur)
    (cond (  (null? guess-lst)
             val  )
          (  (> (cadr (car guess-lst)) max-occur)
             (best-guess1 (cdr guess-lst)
                          (car (car guess-lst))
                          (cadr (car guess-lst)))  )
          (  else
             (best-guess1 (cdr guess-lst)
                          val
                          max-occur)  )))

(define (r-lst->points r-lst)
    (cond (  (null? r-lst) NIL  )
          (  else
             (cons (nth *mem* (entry-index (car r-lst)))
                   (r-lst->points (cdr r-lst)))  )))


;;;
```

```
;;; DISTANCE ROUTINES
;;;

(define (exact first second)
            (cond (  (eq? first second)  0  )
                  (  (or (and (eq? first 'yes) (neq? second 'no))
                         (and (eq? second 'yes) (neq? first 'no)))
                     2  )
                  (  else 10   )))

(define (distance-between dim val-lst1 val-lst2 num1 num2)
    (cond (  (<=0? num1)
             (distance-between dim val-lst1 val-lst2 1 num2)  )
          (  (<=0? num2)
             (distance-between dim val-lst1 val-lst2 num1 1)  )
          (  else
             (distance-between1 dim val-lst1 val-lst2 num1 num2)  )))


(define (distance-between1 dim val-lst1 val-lst2 num1 num2)
    (let (  (metric (table-entry *metric-table* dim))  )
         (cond (  metric
                  (metric val-lst1 val-lst2 num1 num2)  )
               (  (and (null?   (cdr val-lst1))
                       (null?   (cdr val-lst2))
                       (number? (caar val-lst1))
                       (number? (caar val-lst2)))
                  (set metric (table-entry *metric-table*
                                           'numerical-distance))
                  (metric val-lst1 val-lst2 num1 num2)  )
               (  (and (every (lambda (pair) (number? (car pair)))
                              val-lst1)
                       (every (lambda (pair) (number? (car pair)))
                              val-lst2))
                  (set metric (table-entry *metric-table*
                                           'statistical-distance))
                  (metric val-lst1 val-lst2 num1 num2)  )
               (  else
                  (set metric (table-entry *metric-table*
                                           'structured-distance))
                  (metric val-lst1 val-lst2 num1 num2)  ))))

(define (default-numerical-distance val-lst1 val-lst2 num1 num2)
  (let (  (x (caar val-lst1))
          (y (caar val-lst2))  )
       (cond (  (or (not (number? x))
                    (not (number? y)))
                100  )
             (  (and (zero? x) (zero? y))
                0  )
             (  else
                (* 100
                   (/ (abs (->float (- x y)))
                      (max (abs x) (abs y) (abs (- x y)))))  ))))
```

```
;;; for statistical distance, (u1-u2)^2 * (mn/(m+n^2)) * 1/(St^2)
;;; for m=n=1, use (x-y)/(max(x,y,abs(x-y))) or equiv
(define (default-statistical-distance val-lst1 val-lst2 num1 num2)
    (cond (  (or (null? val-lst1) (null? val-lst2))
             100  )
          (  (and (null? (cdr val-lst1)) (null? (cdr val-lst2)))
             (default-numerical-distance val-lst1 val-lst2 num1 num2)  )
          (  else
             (let* (  (stats1 (statistics val-lst1))
                      (stats2 (statistics val-lst2))
                      (stats3 (map + stats1 stats2))  )
                   (* 100
                      (- 1
                         (/ (+ (variance stats1) (variance stats2))
                            (variance stats3)))))  )))

(define (statistics val-lst)
    (iterate *loop (  (n 0) (sum 0) (sum-of-squares 0) (lst val-lst)  )
        (cond (  (null? lst) (list n sum sum-of-squares)  )
              (  else
                 (*loop (+ n (cadar lst))
                        (+ sum (* (caar lst) (cadar lst)))
                        (+ sum-of-squares
                           (* (caar lst) (caar lst) (cadar lst)))
                        (cdr lst))  ))))

(define (variance stats)
    (let (  (n (car stats))
            (sum (cadr stats))
            (sum-of-squares (caddr stats))  )
         (cond (  (zero? n)
                  0  )
               (  else
                  (- sum-of-squares
                     (/ (* sum sum) n))  ))))

;;; take average of percent matches and times by 100
(define (default-structured-distance val-lst1 val-lst2 num1 num2)
    (let* (  (sum1 (num-vals val-lst1))
             (sum2 (num-vals val-lst2))
             (match-count (count-repititions val-lst1 val-lst2))  )
          (cond (  (or (zero? sum1) (zero? sum2))
                   100  )
                (  else
                   (- 100.0
                      (* 50 (+ (/ (car match-count)  sum1)
                               (/ (cadr match-count) sum2))))  ))))

(define (count-repititions val-lst1 val-lst2)
    (iterate *loop (  (match1 0) (match2 0) (lst1 val-lst1)  )
        (cond (  (null? lst1)
                 (list match1 match2)  )
              (  (assq (caar lst1) val-lst2)
                 (*loop (+ match1 (cadar lst1))
                        (+ match2 (cadr (assq (caar lst1) val-lst2)))
```

```
                    (cdr lst1))  )
             (  else
                (*loop match1 match2 (cdr lst1))  ))))

(define (num-vals val-lst)
    (iterate *loop(  (sum-so-far 0) (lst val-lst)  )
        (cond (  (null? lst) sum-so-far  )
              (  else
                 (*loop (+ sum-so-far (cadar lst))
                        (cdr lst))  ))))

(set (table-entry *metric-table* 'structured-distance)
     default-structured-distance)
(set (table-entry *metric-table* 'numerical-distance)
     default-numerical-distance)
(set (table-entry *metric-table* 'statistical-distance)
     default-statistical-distance)

;;;
;;; GENERALIZE
;;;

(define (generalize case-lst)
    (generalize1 (car case-lst) (cdr case-lst)))

(define (generalize1 case1 case-lst)
    (cond (  (null? case-lst) case1  )
          (  (null? case1)
             (generalize1 (car case-lst) (cdr case-lst))  )
          (  else
             (generalize1 (blend-cases case1 (car case-lst))
                          (cdr case-lst))  )))

(define (blend-cases case1 case2)
    (cond (  (null? case1) case2  )
          (  (null? case2) case1  )   ;not needed
          (  else
             (let (  (feature1 (car case1))
                     (feature2 (assq (caar case1) case2))  )
                  (if feature2
                      (cons (blend-features feature1 feature2)
                            (blend-cases (cdr case1)
                                         (del alikev? feature2 case2)))
                      (cons feature1
                            (blend-cases (cdr case1) case2))))  )))

(define (blend-features feature1 feature2)
    (cond (  (null? feature2) feature1  )
          (  (null? feature1) feature2  )
          (  else
             (append (list (car feature1)
                           (max (cadr feature1) (cadr feature2)))
                     (blend-val-lsts (cddr feature1) (cddr feature2)))  )))

(define (blend-val-lsts val-lst1 val-lst2)  ;blend val-lsts
```

```
    (cond (  (null? val-lst1) val-lst2  )
          (  (null? val-lst2) val-lst1  )
          (  else
             (let (  (val1 (car val-lst1))
                     (val2 (ass equal? (caar val-lst1) val-lst2))  )
                  (if val2
                      (cons (blend-values val1 val2)
                            (blend-val-lsts (cdr val-lst1)
                                            (del alikev? val2 val-lst2)))
                      (cons val1
                            (blend-val-lsts (cdr val-lst1) val-lst2))))  )))

(define (blend-values val1 val2)
    (list (car val1) (+ (cadr val1) (cadr val2))))


;;;
;;; HELP for users not knowing about symbols in the database
;;;

(define (help test-obj)
    (map (lambda (feature)
             (cond (  (sub-symbol? test-obj (car feature))
                      (pretty-print feature (standard-output))
                      (newline (standard-output))  )
                   (   (ass sub-symbol? test-obj (cddr feature))
                       (pretty-print (list (car feature)
                                           (ass sub-symbol? test-obj (cddr feature)))
                                     (standard-output))
                       (newline (standard-output))  )
                    (  else T  )))
         (generalize *mem*))
    T)

(define (sub-symbol? obj1 obj2)
    (cond (  (equiv? obj1 obj2) T  )
          (  (or (not (symbol? obj1)) (not (symbol? obj2)))
             NIL  )
          (  (subsetstring (list->string (delq #\- (string->list
                                                    (symbol->string obj1))))
                           (list->string (delq #\- (string->list
                                                    (symbol->string obj2)))))
             T  )
          (  else NIL  )))

(define (subsetstring string1 string2)
    (let (  (index   (string-posq (char string1) string2))
            (length1 (string-length string1)             )
            (length2 (string-length string2)             )  )
         (cond (  (zero? length1) 0  )
               (  (or (null? index)
                      (> length1 (- length2 index)))
                  NIL  )
               (  (string-equal? string1 (substring string2 index length1))
                  index  )
```

```
            (  else
               (nil-sum (add1 index)
                  (subsetstring string1
                                (nthchdr string2 (add1 index))))  ))))

(define (nil-sum num1 num2)
    (cond (  (or (not (number? num1))
                 (not (number? num2)))
             NIL  )
          (  else (+ num1 num2)  )))


;;;
;;; STANDARDIZE
;;;

(define (standardize point)
    (if (template? point)
        (alphabetize (standardize-template point) car)
        (alphabetize (cons (list 'num-cases 1 (list 'foo 1))
                           (standardize0 point) )
                     car) ) )


;; Check and remove duplicates in case the user inputs the exact same
;; feature-value pair more than once.
(define (standardize0 point)
    (remove-dups (standardize1 point 1)) )

(define (standardize-template point)
    (remove-dups (standardize1 point (num-cases-of point))) )


;; Standardizes each feature in a case until none are left.
(define (standardize1 point count)
    (cond (  (null? point) NIL  )
          (  else (cons (standardize-feature (car point) count)
                        (standardize1 (cdr point) count))  )))


;; Dispatches to the appropiate procedure depending on the 'type' of feature
;; being looked at:  if no value list given then goes to CHECK-YES-NO,
;; if the name of the feature is NUM-CASES then off to STANDARDIZE-NUM-CASES,
;; if already standardized then only need to alpha&standardize value list,
;; and if any other then construct {feature-name -1 alphab&stand_val-lst}.
(define (standardize-feature feature count)
    (cond (  (null? feature)  NIL  )
          (  (atom? feature)  (make-yes-no-feature feature count) )
          (  (null? (cdr feature))  (make-yes-no-feature (car feature) count) )
          (  (eq? 'num-cases (car feature))
             (list 'num-cases -1 (standardize-num-cases (cdr feature) count)) )
          (  (and (cddr feature)
                  (number? (cadr feature)))
             (append (list (car feature) (cadr feature))
                     (alphabetize (standardize-val-lst (cddr feature) count)
```

```
                                car))  )
          (  else
             (append (list (car feature) -1)
                     (alphabetize (standardize-val-lst (cdr feature) count)
                                car))  )))


(define (standardize-num-cases lst count)
    (cond ( (null? lst)          (list 'foo 1) )
          ( (number? (car lst))
            (if (equal? count (car lst))
                (list 'foo count)
                (warn 'standardize-num-cases (car lst) count) ) )
          ( (atom? (car lst))
            (warn 'standardize-num-cases (car lst) count) )
          ( else lst) ))


(define (standardize-val-lst val-lst count)
    (cond (  (null? val-lst) NIL  )
          (  (atom? (car val-lst))
             (cons (list (car val-lst) count)
                   (standardize-val-lst (cdr val-lst) count))  )
          (  (null? (cdar val-lst))
             (cons (list (caar val-lst) count)
                   (standardize-val-lst (cdr val-lst) count)) )
          (  (and (number? (cadar val-lst))
                  (null? (cddar val-lst)))
             (cons (car val-lst)
                   (standardize-val-lst (cdr val-lst) count)) )
          (  (number? (cadar val-lst))
             (cons (list (caar val-lst) (cadar val-lst))
                   (standardize-val-lst (cdr val-lst) count)) )
          (  else
             (warn 'standardize-val-lst (car val-lst))
             (cons (list (caar val-lst) count)
                   (standardize-val-lst (cdr val-lst) count)) )))

;; Creates a default value-list {YES 1} if feature comes with no value attached
;; and name is not prefixed with "NO" or "NOT".
(define (check-yes-no boolean-dim)
   (destructure* (  ( stringname    (symbol->string boolean-dim   ))
                    ( (name neg?)   (find-prefix "NOT-" stringname))
                    ( (name2 neg2?) (find-prefix "NO-" stringname ))  )
                 (cond (  neg?
                          (return (string->symbol name) 'no)  )
                       (  neg2?
                          (return (string->symbol name2) 'no)  )
                       (  else
                          (return boolean-dim 'yes)  ))))


;;;
```

```
;;; UTILITIES
;;;

(define (square num)
    (* num num))

(define (round num accuracy)
  (let ( (power (expt 10 accuracy)) )
    (->float (divide (->integer (+ .5 (* num power))) power)) ))

(define (multiple-entry-print lst port)
      (for (elt in lst)
        (do (pretty-print elt port)
            (newline port)
            (newline port) )))

(define (make-yes-no-feature feature count)
    (receive (name yes/no)                          ; local vars
             (check-yes-no feature)                 ; returns two values
             (list name -1 (list yes/no count)) ) ) ; body

(define (template? pt)
    (any (lambda (feature)
           (if (pair? feature)
               (eq? 'num-cases (car feature))
               (eq? 'num-cases feature) ) )
         pt))

(define (num-cases-of point)
  (labels
    ( ((sum-lst lst) (sum-lst1 lst 0))
      ((sum-lst1 lst psum)
       (if (null? lst) psum (sum-lst1 (cdr lst) (add (car lst) psum))) )
      ((dispatch pt)
       (cond
          ( (or (null? pt) (null? (car pt)))  1)
          ( (atom? (car pt))
            (num-cases-of (cdr pt)) )
          ( (eq? 'num-cases (caar pt))
            (cond ( (null? (cdar pt)) 1)
                  ( (cddr (car pt)) =>
                    (lambda (x) (sum-lst (map cadr x))) )
                  ( else                                    ; (NUM-CASES val)
                    (cadr (car pt)) ) ))                    ; return val
          (else (num-cases-of (cdr pt))) ) )
    (dispatch point) ))


(define (name-of point)
   (let ( (name (val-lst-of point *named-dim*)) )
        (cond (  name name  )
              (  (zero? (num-cases-of point))
                 NIL  )
              (  else
                 (list (list '? (num-cases-of point)))  ))))
```

```
(define (index-of name)
        (set *remindings* (fetch-r-lst *current*))
        (index-of1 name *remindings*))

(define (index-of1 name r-lst)
   (cond (  (null? r-lst) NIL  )
         (  (assq name (entry-name (car r-lst)))
            (entry-index (car r-lst))  )
         (  else
            (index-of1 name (cdr r-lst))  )))

(define (find-prefix prefix name)
   (let (  (prelen (string-length prefix))  )
        (cond (  (<= (string-length name) prelen)
                 (list name NIL)  )
              (  (string-equal? prefix (string-slice name 0 prelen))
                 (list (string-nthtail name prelen) T)  )
              (  else
                 (list name NIL)  ))))


(define (call-to-mind dim value)
    (call-to-mind1 dim value *mem*))

(define (call-to-mind1 dim value mem-lst)
    (cond (  (null? mem-lst) NIL  )
          (  (assq value (val-lst-of (car mem-lst) dim))
             (cons (car mem-lst)
                   (call-to-mind1 dim value (cdr mem-lst)))  )
          (  else (call-to-mind1 dim value (cdr mem-lst))  )))

(define (val-lst-of point dim)
    (cddr (assq dim point)))

(define (collapse value-lst)
    (cond ( (null? value-lst) NIL )
          ( (atom? (car value-lst))
            (cons (car value-lst) (collapse (cdr value-lst))) )
          ( else
            (cons (caar value-lst) (collapse (cdr value-lst))) )))


(define (warn procedure-name . parameters)
    (format T "~%!WARNING! in procedure ~s:~%" procedure-name)
    (cond (  (eq? procedure-name 'standardize-val-lst)
             (format T
                     "   >> Value ~s cannot be a list of values~%"
                     (car parameters))
             (format T
                     "   >> ~s being used as value instead~%"
                     (caar parameters))  )
          (  (eq? procedure-name 'standardize-num-cases)
             (format T
                     "   >> Number of cases ~s should equal ~s~%"
```

```
                (car parameters) (cadr parameters))
            (format T
                "   >> ~s being used.~%"
                (cadr parameters)) )
         (  (eq? procedure-name 'entropy-function)
            (format T
                "   >> Attempting to find entropy of ~s~%"
                (car parameters))
            (format T
                "   >>   when *max-entropy* is ~s~%"
                (cadr parameters))  )
         (  else
            T  ))
    (newline (standard-output)))

(define (remove-dups lst)
   (reverse (remove-dups1 lst '())))

(define (remove-dups1 duplst result)
   (cond (  (null? duplst) result  )
         (  (mem? alikev? (car duplst) result)
              (remove-dups1 (cdr duplst) result)  )
         (  else (remove-dups1 (cdr duplst)
                               (cons (car duplst) result))  )))

(define (remove-elt elt lst)
   (reverse (remove-elt1 elt lst '())))

(define (remove-elt1 elt with-elts no-elts)
   (cond (  (null? with-elts) no-elts  )
         (  (eq? elt (car with-elts))
            (remove-elt1 elt (cdr with-elts) no-elts)  )
         (  else (remove-elt1 elt (cdr with-elts)
                   (cons (car with-elts) no-elts))  )))

;; Alphabetize routines:  ALPHABETIZE is called by STANDARDIZE and
;; the result is an alphabetized feature list for each case.
;; Note numbers are defined to come *after* all symbols lexigraphically.

(define (alpha-before? one two)
    (cond (  (number? one)
             (if (and (number? two) (< one two))
                 T
                 NIL)  )
          (  (number? two) T  )
          (  else
             (let* (  (first (string->list (symbol->string one)))
                      (second (string->list (symbol->string two)))  )
                   (alpha-before2? first second))  )))

(define (alpha-before2? first second)
   (let (  (first-code (if (null? first) NIL (char->ascii (car first))))
           (second-code (if (null? second)
                            NIL
                            (char->ascii (car second))))  )
```

```
        (cond (  (null? first) T  )
              (  (null? second) NIL  )
              (  (< first-code second-code) T  )
              (  (> first-code second-code) NIL  )
              (  else (alpha-before2? (cdr first) (cdr second))  ))))

(define (alpha-insert obj lst proc)
   (let (  (name (proc obj))  )
        (cond (  (null? lst) (list obj)  )
              (  (null? obj) lst  )
              (  (alpha-before? name (proc (car lst)))
                 (cons obj lst)  )
              (  else
                 (cons (car lst) (alpha-insert obj (cdr lst) proc))  ))))

(define (alpha-list orig-list list-so-far proc)
   (cond (  (null? orig-list) list-so-far  )
         (  (null? list-so-far)
            (alpha-list (cdr orig-list) (list (car orig-list)) proc)  )
         (  else
            (alpha-list (cdr orig-list)
                        (alpha-insert (car orig-list) list-so-far proc)
                        proc)  )))

(define (alphabetize lst proc)
   (alpha-list lst '() proc))

;;;
;;; INITIALIZATION
;;;


(define (clean)
    (set *new-one* NIL)
    (set *current* NIL)
    (set *conclusions* NIL)
    (set *remindings* NIL)
    (set *mem* NIL)
    (reset)
    )


(define (tidy)
    (set *new-one* NIL)
    (set *current* NIL)
    (set *conclusions* NIL)
    (set *remindings* NIL)
    (reset)
    )

(define (delete-last-memory)
     (cond (  (null? *mem*)  NIL  )
           (  else
              (format T "~%The following case is being deleted:~%")
              (pp (car *mem*))
```

```
            (newline (standard-output))
            (set *mem* (cdr *mem*))
            repl-wont-print )))

(define (block-write exp val port)
    (write-line port "(block" )
    (writes     port "(lset " )
    (write      port exp)
    (writes     port " ")
    (if (not (number? val))
        (write-line port "(quote " )
        T)
    (pretty-print val port)
    (newline port)
    (if (not (number? val))
        (writes port ")" )
        T)
    (write-line port ")" )
    (writes port "(quote " )
    (write port exp)
    (write-line port ")" )
    (write-line port ")" )
    (newline port)
    (newline port)
    (writes (standard-output) "Done with: ")
    (write (standard-output) exp)
    (newline (standard-output)))

(define (mem-dump file)
    (cond (  (file-exists? file)
             (format T "~%File ~s already exists.  Use another name.~%"
                   file)  )
          (  else
             (with-open-ports (  (port (open file '(out)))  )
                 (writes port "(herald \""              )
                 (write port file                       )
                 (write-line port "\")"                 )
                 (write-line (standard-output) "...HERALD")
                 (newline port                          )
                 (newline port                          )
                 (block-write '*mem*          *mem*          port)
                 (block-write '*max-entropy*  *max-entropy*  port)
                 (block-write '*named-dim*    *named-dim*    port)
                 (block-write '*special-dim*  *special-dim*  port)
                 (write-line port "'BYE"                  ))  )))
```

Appendix B: Clustering Algorithm

MU uses the routine *CLUSTER* to find a natural break point in a list of cases, dividing that list into a "close" group $\mathcal{C}$ and a "distant" group $\mathcal{D}$. Referring to a list of numerical distances, the algorithm attempts to partition the list into two groups x_i and y_j such that the sum of the squared deviations *within* the groups is locally minimized. That is, *CLUSTER* attempts to find a local minimum for

$$\sum_{i=1}^{m}(x_i - \bar{x})^2 + \sum_{j=1}^{n}(y_j - \bar{y})^2.$$

Computationally, however, this calculation is inefficient. Instead, we can note that if μ represents the mean value of all the distances x_i and y_j combined, then

$$\begin{aligned}
\sum_{i=1}^{m}(x_i - \mu)^2 &= \sum_{i=1}^{m}\left(x_i - \frac{m\bar{x} + n\bar{y}}{m+n}\right)^2 \\
&= \sum_{i=1}^{m}\left(x_i - \bar{x} + \frac{n}{m+n}(\bar{x} - \bar{y})\right)^2 \\
&= \sum_{i=1}^{m}(x_i - \bar{x})^2 + \frac{2n}{m+n}(\bar{x} - \bar{y})\sum_{i=1}^{m}(x_i - \bar{x}) + \frac{mn^2}{(m+n)^2}(\bar{x} - \bar{y})^2 \\
&= \sum_{i=1}^{m}(x_i - \bar{x})^2 + \frac{mn^2}{(m+n)^2}(\bar{x} - \bar{y})^2
\end{aligned}$$

Similarly,

$$\sum_{j=1}^{n}(y_j - \mu)^2 = \sum_{j=1}^{n}(y_j - \bar{y})^2 + \frac{m^2n}{(m+n)^2}(\bar{y} - \bar{x})^2$$

Combining these results gives

$$\sum_{i=1}^{m}(x_i - \mu)^2 + \sum_{j=1}^{n}(y_j - \mu)^2 = \sum_{i=1}^{m}(x_i - \bar{x})^2 + \sum_{j=1}^{n}(y_j - \bar{y})^2 + \frac{mn}{(m+n)}(\bar{x} - \bar{y})^2$$

Since the quantity on the left is constant for the given list of distances no matter what the partition, we can minimize the total intra-group summed squared deviations simply by choosing our partition to find the first maximum for the last quantity,

$$\frac{mn}{(m+n)}(\bar{x}-\bar{y})^2.$$

This will give us the first clean break in the distance list.

Appendix C: Evocativeness Algorithm

EVOCATIVENESS attempts to determine how strongly a presenting case τ_0 brings to mind a definite diagnosis. What we would really like to measure is how much *information* about the diagnosis we gain from τ_0. Does τ_0 strongly suggest only one diagnosis, or does it bring to mind ten diagnoses which are equally likely? One way to measure this information content I is to relate it to the entropy D of a probability distribution.

Let the total number of diagnoses found in the top-cluster $\mathcal{Q}$ be T. Assuming that the probability p_i of diagnosis i being correct is proportional to the number of times n_i it occurs in the top-cluster, we can calculate the entropy (disorder) of the distribution:

$$\begin{aligned} D &= -\sum_i p_i \ln p_i \\ &= -\sum_i \frac{n_i}{T} \ln \frac{n_i}{T} \\ &= -\frac{1}{T}\sum_i n_i \ln n_i + \frac{\ln T}{T}\sum_i n_i \\ &= -\frac{1}{T}\sum_i n_i \ln n_i + \ln T \end{aligned}$$

The entropy function D ranges from a value of 0, occurring when only one diagnosis is represented, to $\ln N$, where N represents the total number of possible diagnoses in the memory $\mathcal{M}$. We can scale the entropy to range of 0 to 1 simply

by dividing by $\ln N$:

$$S = \frac{1}{\ln N}\left(-\frac{\sum_i n_i \ln n_i}{T} + \ln T\right)$$

We can further adjust the scale so that an scaled-entropy of 0 corresponds to the maximum evocativeness allowed by the system, while a scaled-entropy of 1 corresponds to the minimum evocativeness allowed by the system. This is the final evocativeness number E used by MU.

$$E = S\,(*\texttt{min} - \texttt{evoc}*) + (1 - S)\,(*\texttt{max} - \texttt{evoc}*)$$

References

[1] Shortliffe EH, Buchanan BG, Feigenbaum EA. Knowledge engineering for medical decision making: a review of computer-based clinical decision aids. *Proceedings of the IEEE*, 67:1207–1224; 1979.

[2] Michie D. *On Machine Intelligence.* Edinburgh, Edinburgh University Press, 1974; cited in Kulikowski CA. Artificial intelligence methods and systems for medical consultation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-2 (5):464–476; 1980.

[3] Turing A. "Computing machinery and intelligence," 1950, in Feigenbaum EA and Feldman J (editors). *Computers and Thought*, New York, McGraw-Hill, 1963.

[4] Winston PH. *Artificial Intelligence*, 2^{nd} edition. Reading, Massachusetts, Addison Wesley Publishing Company. 1984.

[5] McCorduck P. *Machines Who Think.* San Francisco, W.H. Freeman and Company. 1979.

[6] Barr A, Feigenbaum EA, Cohen P (editors). *Handbook of Artificial Intelligence*, 3 volumes. Los Altos, California, William Kaufmann, 1981–1982.

[7] Mason VR. Sickle cell anemia. *JAMA*, 79:1318–1320; 1922.

[8] Johnson CS. Sickle cell anemia. *JAMA*, 254(14):1958–1963; 1985.

[9] Shortliffe EH. *Computer-Based Medical Consultations: MYCIN.* New York, American Elsevier Publishing Co., 1976.

[10] Shortliffe EH, Axline SG, Buchanan BG, Merigan TC, Cohen SN. An artificial intelligence program to advise physicians regarding antimicrobial therapy. *Computers and Biomedical Research*, 6:544-560; 1973.

[11] Pauker SG, Gorry GA, Kassirer JP, and Schwartz WB. Towards the simulation of clinical cognition: taking a present illness by computer. *American Journal of Medicine*, 60:981–996; 1976.

[12] Szolovits P, Pauker SG. Categorical and probalistic reasoning in medical diagnosis. *Artificial Intelligence*, 11:115–144; 1978.

[13] Minsky M. A framework for representing knowledge. In Winston PH (ed), *The Psychology of Computer Vision*, New York, Mc-Graw Hill, 1975.

[14] Miller RA, Pople HE, Myers JD. INTERNIST-1, an experimental computer-based diagnostic consultant for general internal medicine. *New England Journal of Medicine*, 307:468–476; 1982.

[15] Barnett GO, Cimino JJ, Hupp JA, Hoffer EP. DXplain: an evolving diagnostic decision-support system. *JAMA*, 258(1):67–74; 1987.

[16] Bush V. As we may think. *Atlantic Monthly*, July 1945, pp 101–108.

[17] Koss N, Feinstein AR. Computer-aided prognosis. *Archives of Internal Medicine*, 127:448–459; 1971.

[18] Feinstein AR, Rubinstein JF, Ramshaw WA. Estimating prognosis with the aid of a conversational-mode computer program. *Annals of Internal Medicine*, 76:911–921; 1972.

[19] Rosati RA, McNeer Jf, Starmer CF, Mittler BS, Morris JJ, Wallace AG. A new information system for medical practice. *Archives of Internal Medicine*, 135:1017–1024; 1975.

[20] Fries JF. Time-oriented patient records and a computer databank. *JAMA*, 222(12):1536–1542; 1972.

[21] Fries JF, McShane DJ. ARAMIS (The American Rheumatism Association Medical Information System), a prototypical national chronic-disease data bank. *Western Journal of Medicine*, 145:798–804; 1986.

[22] Fries JF. A data bank for the clinician? *New England Journal of Medicine*, 294(25):1400–1402; 1976.

[23] Diamond GA, Staniloff HM, Forrester JS, Pollock BH, Swan HJC. Computer-assisted diagnosis in the noninvasive evaluation of patients with suspected coronary artery disease. *Journal of the American College of Cardiology*, 1(2):444-455; 1983.

[24] Lee KL, Pryor DB, Harrell FE, et al. Predicting outcome in coronary disease: statistical models versus expert clinicians. *American Journal of Medicine*, 80:553–560; 1986.

[25] Kulikowski CA. Pattern recognition approach to medical diagnosis. *IEEE Transactions on Systems Science and Cybernetics*, SSC-6(3):173–178; 1970.

[26] Stanfill C, Waltz D. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[27] Kolodner JL. Using experience in clinical problem solving. *Proc. AIM 87*, July 1987.

[28] Gelernter D, Sklar M. Machine musing: preliminary report, in a psychiatric domain. *Proceedings AAMSI Congress 86*. May, 1986.

[29] Fertig S, Gelernter D. Maven the muser. *Proc. AIM 87*, July 1987.

[30] Gelernter D, Sklar M. *Machine Musing and the Smart Notepad.* Yale University Department of Computer Science Technical Report, TR-466, March 1986.

[31] Miller PL. Critiquing: a different approach to expert computer advice in medicine. *Proc. Eighth SCAMC*, November 1984 pp 17–23.

[32] Miller PL. Critiquing anesthetic management: the "ATTENDING" computer system. *Anesthesiology*, 58:362–369, 1983.

[33] Miller PL, Black HR. Medical plan-analysis by computer: critiquing the pharmacologic management of essential hypertension. *Computers and Biomedical Research*, 17:38–54, 1984.

[34] McSherry DMG. Intelligent dialogue based on statistical models of clinical decision-making. *Statistics in Medicine*, 5:497–502; 1986.

[35] Teach RL, Shortliffe EH. An analysis of physician attitudes regarding computer-based clinical consultation systems. *Computers and Biomedical Research*, 14:542–558, 1981.

[36] Rees JA, Adams NI. "T: a dialect of Lisp or, lambda: the ultimate software tool." In *Proceedings of the 1982 ACM Symposium on Lisp and Functional Programming.* Association for Computing Machinery, 1982.

[37] Rees JA, Adams NI, Meehan JR. *The T Manual*, 4^{th} edition. Yale University Computer Science Department, 1984.

[38] Winston PH, Horn B. *LISP.* Reading, Massachusetts, Addison Wesley Publishing Company. 1981.

[39] Steele GL, Sussman GJ. The revised report on Scheme, a dialect of Lisp. AI Memo 452, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1978.

[40] Hamilton M. A rating scale for depression. *J. Neurol. Neurosurg. Psychiat.*, 23:56–62; 1960.

[41] Hamilton M. Development of a rating scale for primary depressive illness. *Brit. J. Soc. Clin. Psychology*, 6:278–296; 1967.

[42] Phillips DS. *Basic Statistics for the Health Science Students.* New York, W.H. Freeman, 1978.

[43] Robbins SL, Cotran RS, Kumar V. *The Pathologic Basis of Disease*, 3^{rd} edition. Philadelphia, W.B. Saunders, 1984.

[44] Schwartz WB, Patil RS, Szolovits P. Sounding board. Artificial intelligence in medicine: where do we stand? *New England Journal of Medicine*, 316(11):685–688; 1987.

YALE MEDICAL LIBRARY
3 9002 01012 9972
ALE MEDICAL

YALE MEDICAL LIBRARY

Manuscript Theses

Unpublished theses submitted for the Master's and Doctor's degrees and deposited in the Yale Medical Library are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but passages must not be copied without permission of the authors, and without proper credit being given in subsequent written or published work.

This thesis by has been used by the following persons, whose signatures attest their acceptance of the above restrictions.

NAME AND ADDRESS	DATE